Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL FINAL DE GRAU

TFG TITLE: Time to land prediction in Barcelona-El Prat airport based on machine learning classification models.

DEGREE: Grau en Enginyeria d'Aeronavegació

AUTHOR: Víctor Mouriño Pérez

ADVISOR: Cristina Barrado Muxí

DATA: July 24th, 2020

Título: Predicción del tiempo de aterrizaje en el aeropuerto de Barcelona-El Prat a partir de modelos de clasificación de aprendizaje automático.

Autor: Víctor Mouriño Pérez

Directora: Cristina Barrado Muxí

Fecha: 24 de julio de 2020

Resumen

Este documento contiene un método para estimar el tiempo de aterrizaje en el aeropuerto de Barcelona-El Prat mediante modelos de clasificación de aprendizaje automático. El objetivo del proyecto, por lo tanto, no es predecir una variable continua sino si el tiempo de aterrizaje cae dentro de una de las siguientes categorías: avanzado, planeado o retrasado. Adicionalmente, se han añadido dos categorías llamadas muy avanzado y muy retrasado que contienen aquellos vuelos que han tenido valores anómalos en su tiempo de aterrizaje, ya sea por ir muy rápido o muy lento.

Para obtener los datos, se ha usado una antena ADS-B ubicada en el tejado de la escuela de ingeniería y aeroespacial de Castelldefels. Esta antena capta las señales de todas las llegadas a Barcelona que posteriormente se decodifican gracias a un programa ya existente escrito en C#. Mediante un programa propio escrito en python, se han extraído las características más relevantes de estos vuelos y se han presentado en forma de matriz con el objetivo de que fueran entendibles por los distintos modelos.

Los datos han sido escalados y divididos en muestras de entrenamiento y de test. Las primeras utilizadas para que los distintos modelos aprendan y las segundas para comprobar la eficacia de los mismos.

Se han entrenado seis modelos distintos, cuatro de ellos han obtenidos unos valores de exactitud por encima del 60%, otro de ellos ha llegado a valores del 75% y otro no ha logrado pasar del 30%. Los tres mejores, se han ajustado mediante dos técnicas distintas: una búsqueda aleatoria y una búsqueda de cuadrícula. Se ha podido comprobar que la búsqueda aleatoria es mucho mejor puesto que se obtienen los mismos resultados en mucho menos tiempo y se requiere mucha menos capacidad de computación. Además, se han utilizado también distintos métodos para mejorar los resultados de los modelos ya ajustados como clasificadores de votación o impulsores.

Finalmente, se han implementado técnicas de sobremuestreo para el problema de las cinco categorías puesto que los casos extremos están muy infrarrepresentados. Gracias a estas técnicas, se ha podido mejorar la exactitud específica de estas dos categorías, pero a cambio se ha perdido exactitud global.

El rendimiento de estos modelos, se podría mejorar en el futuro añadiendo más vuelos a las muestras de datos o añadiendo más características de los mismos.

Title: Time to land prediction in Barcelona-El Prat airport based on machine learning classification models.

Author: Víctor Mouriño Pérez

Director: Cristina Barrado Muxí

Date: July 24th, 2020

Overview

This document contains a method to assess the landing time in Barcelona-El Prat airport using machine learning classification models. The goal of this project is not to predict a continuous variable but if the time to land falls inside one of the following categories: advanced, delayed or planned. Additionally, two categories called very advanced and very delayed have been included containing flights that have had abnormal values on their time to land, either because of very fast approaches or very slow ones.

To obtain the data, an ADS-B antenna located on the aerospace and telecommunications engineering school of Castelldefels rooftop has been used. This antenna, captures the signals of all the arrivals into Barcelona which are later decoded thanks to an already existing program written in C#. By means of a custom program written in python, the most relevant characteristics of these flights are extracted and presented in a matrix format so that the different models can understand them.

All the data has been scaled and divided in training and test samples. The first ones are used to teach the different models and the second ones are used to measure their efficiency.

Six different models have been trained, four of them reached accuracy values over 60%, another one reached a value of 75% and another one could not go over 30%. The best three models, have been adjusted with two different techniques: random search and grid search. It has been verified that a random search is significantly better since the same results can be obtained but it requires much less time and computing resources. Also, different methods to enhance the results of the already adjusted models such as voting classifiers and boosters have been used.

Finally, oversampling techniques have been implemented to solve the five categories problem as the extreme cases are very underrepresented. Thanks to this technique, the accuracy of these two categories was improved but in turn the overall accuracy descended.

The performance of these models could be upgraded in the future if more flights or more characteristics of those flights were added.

A mis abuelos, Joaquín y Filomena.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ADS-B	Automatic Dependant Surveillance Broadcast
AIP	Aeronautical Information Publication
ARP	Aerodrome Reference Point
ATC	Air Traffic Controller
ANN	Artificial Neural Network
CTA	Control Area
CV	Cross Validation
FIR	Flight Information Region
FN	False Negatives
FP	False Positives
GS	Ground Speed
IAS	Indicated Airspeed
IF	Intermediate Fix
ILS	Instrumental Landing System
ICAO	International Civil Aviation Organization
KT	Knots
LVP	Low Visibility Procedures
METAR	Meteorological Terminal Air Report
ML	Machine Learning
MLP	Multilayer Perceptron
MRS	Minimum Radar Separation
MTOW	Maximum Take-off Weight
OvO	One versus One
OvR	One versus the Rest
SKLEARN	Scientific Kit Learning
SMOTE	Synthetic Minority Over-Sampling Technique
STAR	Standard Arrival Route
SVC	Support Vector Classifier
TMA	Terminal Manoeuvring Area
TN	True Negatives
TP	True Positives
WTC	Wake Turbulence Category

ACKNOWLEDGMENTS

First, I’d like to thank my advisor Cristina Barrado Muxí for her dedication and guidance throughout this project. Without her help it would have been impossible to complete it. During the last six months and despite of the exceptional situation due to the state of alarm, Cristina has always found time to arrange online meetings and keep a good track of the project. When I started in February 2020, my knowledge about machine learning was almost null but she has been a kind mentor and has provided me with all the necessary tools and guidelines to successfully complete this endeavour.

Second, I would like to express my gratitude to my family and friends. During this four-year journey they have been an exceptional help, very necessary in the difficult moments.

Last but not least, I would like to thank Marcos Pérez Batlle for his ADS-B decoder and Daniel García-Monteavaro, Jaume Assens and Francesc Fernandez for their effort in obtaining the aeronautical charts.

INTRODUCTION

The European aerial network regulator (Eurocontrol), estimates that air travel will increase at a constant rate of 1.9% per year. This means that by 2040 the annual number of operations will have increased a 53% and the network will have to allocate more than 16 million extra flights each year. In this scenario, it will be of the upmost importance to develop effective tools that can make correct predictions so that the whole network can work smoothly.

Among the different parameters to predict, time is probably the most important one as it is the key to synchronization. There are already methods to estimate the times at the airport, which include taxi time calculations or the time at which a plane should be ready to start its pushback. Anyway, this problem is far more difficult when planes are airborne, especially during the approach.

At cruise level, aircraft usually follow airways or prescribed tracks and they do it at a constant speed. Hence, calculating the time in which a plane will reach a certain waypoint is relatively easy. On the contrary, during the descent phase and the approach, the speed and trajectory variability are significantly increased specially because of human intervention. Air traffic controllers usually allow aircraft to take shortcuts, or instruct them to hold at certain points but there is not a specific criterion. At the same time, pilots continuously change the speed of aircraft depending on the specific flight conditions. Thus, it is very complicated to predict the time a plane will take to land with a classical algorithm as it is almost impossible to correctly contemplate all the variables affecting the problem.

In this project, a method to calculate the landing time based on machine learning is proposed. Thanks to this technology, it is possible to create models that can account for tens or hundreds of variables and manage highly nonlinear data simplifying the approach to the problem.

Chapter one provides a theoretical background about machine learning. It contains a little explanation of the different models that have been used, how to split and prepare the dataset, how to evaluate and improve the different models and how to assess their performance.

Chapter two explains all the process that was followed to gather, process and present the data in an adequate format. The variables that influence the project are presented and their overall importance is discussed. Also, it is explained how the time to land is converted into categories.

Chapter three presents the main results; the performance of the different models is computed and it is explained how they were improved. Also, the conclusions of the project are exposed and some ideas to improve the performance in the future are given.

The code for this project, can be found in the following link:
https://github.com/VictorMourinoPerez/TFG-Victor-Mourino-Perez

CHAPTER 1. MACHINE LEARNING

Machine learning (ML) is a branch of computer science that aims at creating systems that can learn autonomously. Traditionally, the only way for a computer program to do something was by writing an algorithm that defined the context and the details of every action. This technology, flips the side of the coin, because it is able to do its own calculations and to extract its own conclusions that finally result in a new algorithm that can understand the input data. This kind of systems reduce considerably human intervention and they are much better at dealing with huge amounts of information.

A ML model is just the file trained to recognize certain types of patterns which is eventually capable of creating its own rules. These models require huge quantities of data as an input. The input is usually a matrix and it has to contain a high number of instances (rows) but also a considerable number of features (columns). Instances are the different objects from which models will learn and features describe how the different instances are [1].

There are three main ML problems: classification, regression and clustering problems. Classification problems, like this one, try to predict categories or classes. That means that we will not predict a continuous function, instead we will focus on predicting if the time to land of a plane belongs to one of the following three categories: planned, advanced or delayed. Hence, this project can be defined as a multioutput categorisation problem of three classes.

Also, we will look for “extreme” cases that will be classified as very advanced or very delayed. Adding these two categories will test if the different ML models are able to predict very rare scenarios as not many data will be available.

1.1 Scientific kit learning (sklearn)

Sklearn is a free software ML library for python and it allows users to solve the three different types of ML problems. It also includes functionalities that permit dimensionality reduction and data postprocessing. All the models in the library admit something called grid search or parameter tuning which is used to increase their overall performance [2].

In combination with sklearn, we also used other open source libraries such as pandas and NumPy. The first one is mostly used for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables [3]. NumPy, on the other hand, adds support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays [4].

1.2 ML models

In this section, we briefly describe the different models that have been used in the project, where to find them in the sklearn environment and what are the fundamental principles that govern the decision-making process.

1.2.1 Random forest classifiers

The basic idea behind a random forest classifier is to create an ensemble of decision trees, training them at the same time and outputting the class that is the mode of the predicted classes [5]. The features used at each tree are completely random, thus, a single tree is not a very accurate classifier but combining many of them will result in a good predictor. This phenomenon is called wisdom of the crowd and it states that in most cases, the aggregated answer of many people is better than the opinion of an expert. Random forests are available in sklearn under the ensemble module [6].

Fig. 1.1 illustrates the working principle of a n-trees random forest classifier.

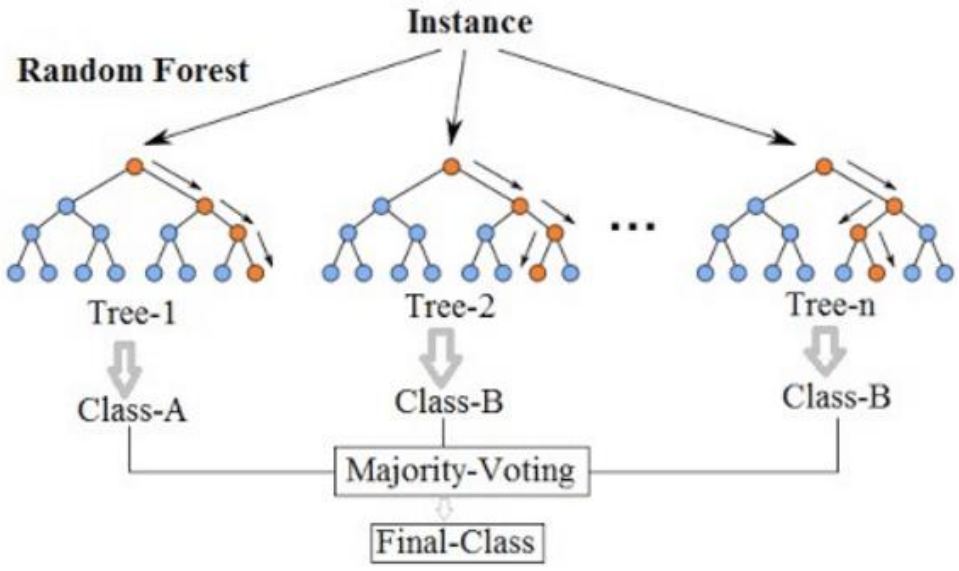


Fig. 1.1 Functioning scheme of a random forest classifier **[7]**.

1.2.2 Logistic regressor

A logistic regressor is a classification method that uses a logistic function to model a binary dependent variable [8]. As it can be seen in Fig. 1.2, the values over the sigmoidal curve are classified as true samples and the values under the curve are classified as false samples.

To use a logistic regressor in multioutput problems like this one, a strategy called one versus the rest (OvR) must be used. Instead of training a single classifier for all the classes, we create one classifier per class. Each individual classifier, will guess if an instance belongs to their class or not and it will also output a confidence score. If two different classifiers think that the same instance belongs to their class, the confidence score is used to break the tie. Logistic regressors can be found in the linear model module of sklearn using the linear regression function [9].

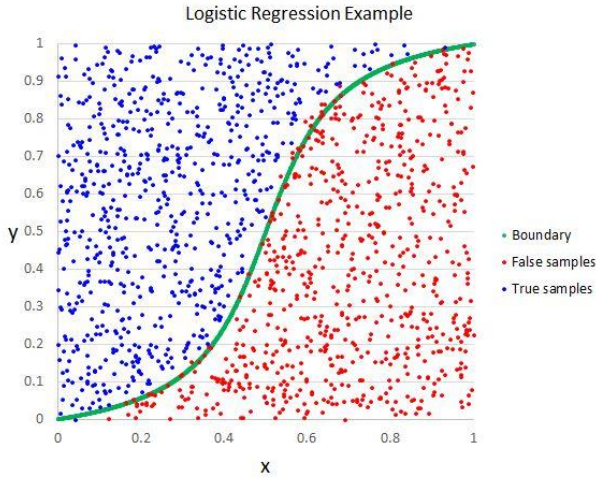


Fig. 1.2 Example of a logistic regression **[10]**.

1.2.3 MLP classifier

A multilayer perceptron (MLP) is a feedforward artificial neural network (ANN). They are composed of at least three layers: an input layer, a hidden layer and an output layer as it can be seen in Fig. 1.3 [11].

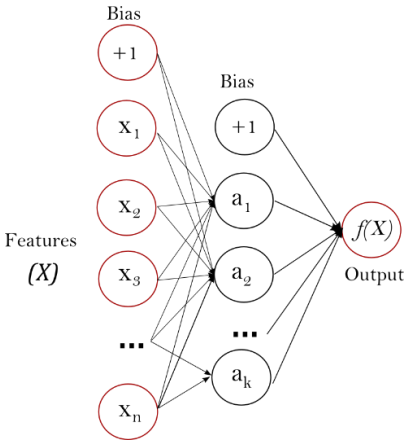


Fig. 1.3 MLP classifier working principle **[12]**.

The hidden layers in any MLP are composed of nodes. Each node has an activation threshold and the functions to calculate their value are highly non-linear. It is also worth mentioning that MLPs are fully connected, that means that every node is connected to all the nodes of the following layer.

This classifier learns by using the least mean square method, comparing the expected output with the output of the classifier. On each iteration, the weights of the connections of the different nodes are changed until the value of the output function is satisfactory or the maximum number of iterations is reached. MLP classifiers can be found under the neural network module of sklearn [13].

1.2.4 K-nearest neighbors

K-nearest neighbors is one of the simplest classifiers in ML but yet it is very effective. The easiest way to understand how this model works is by looking at the example in Fig. 1.4. If we want to classify the green circle as a blue square or a red triangle, the only thing that we have to do is defining the number of neighbors. If K equals 3, then the neighbors of the circle are two triangles and one square, thus the green circle is classified as a triangle. On the other hand, if K equals 5, the neighbors are three squares and two triangles and the circle is classified as a square.

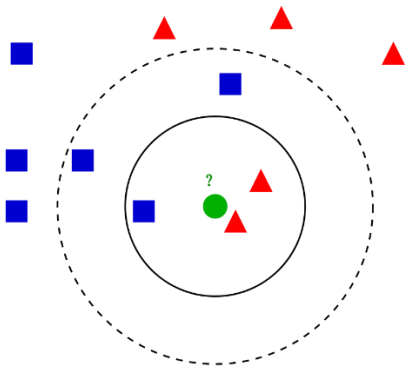


Fig. 1.4 Decision scheme of a K-nearest neighbors **[14]**.

A common strategy that is used with this model is to assign weights to the neighbors equal to the inverse of their Euclidean distance with the instance we are trying to classify. Anyway, the concept of “distance” cannot be clearly visualized for more than three dimensions.

A little drawback of using this method is that it consumes a lot of memory and CPU resources, hence it should not be used with big datasets [15]. It can be found in the sklearn neighbors module [16].

1.2.5 Support vector classification (SVC)

A support vector classifier is very similar to a logistic or a linear regressor, the main difference is that in this case, the model will try to find the optimal hyperplane which is dividing the problem variables. Support vectors are the points near the hyperplane which influence its position and orientation. A hyperplane in a bidimensional space is just a line, thus Fig. 1.5 represents what an SVC would do with two features although it is impossible to imagine how a hyperplane looks like for more than three features [17].

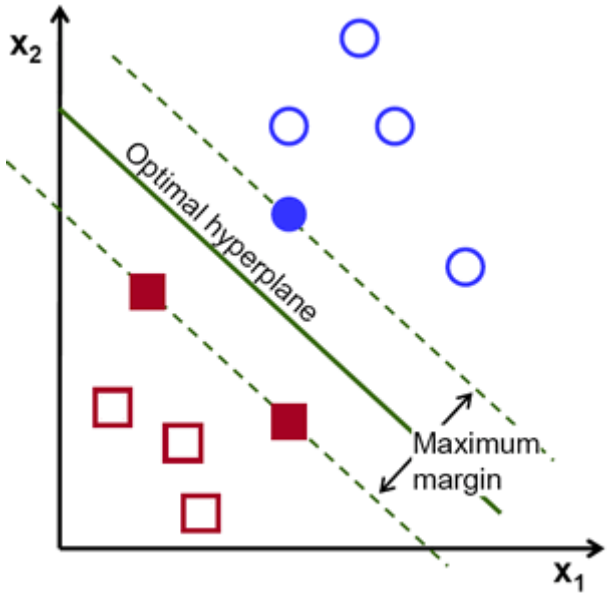


Fig. 1.5 Graphical representation of the optimal hyperplane **[18]**.

This classifier, like the logistic regressor, is meant to work with binary problems only. To use it as a multiclass classifier a slightly different approach is used, instead of using an OvR strategy, we will use a one versus one (OvO). In our project, this means that it will create a classifier to distinguish between advanced and planned flights, another one to distinguish between advanced and delayed flights and a last one to distinguish between planned and delayed flights (for the three categories approach). Generally, this strategy will result in training more classifiers than a simple OvR approach but it is preferred since the classifiers must be only trained in a small part of the dataset and the process is faster. SVC can be found in the sklearn SVM module [19].

1.2.6 Naive Bayes

The Naive Bayes classifier is of course based on the Bayes probability theorem and it is said to be Naive because it assumes that the different features are independent from each other which is false in many cases.

Particularly, we have used a model called complement Naive Bayes (complementNB) under the Naive Bayes sklearn module [20]. This model converts the features of the dataset in a probability vector by counting how many times a certain feature appears and diving it by the total number of samples in the dataset. Once the features probabilities are calculated, the model applies the Bayes theorem and it computes the probability of a certain class happening. The class with the highest probability is the output class.

1.3 Training and test set

The input data of any machine learning model has to be split into a training set and a test set. The training set will be used to teach the models, that means that they will try to understand why all the features result in a time to land being advanced, planned or delayed. On the other hand, the test set will be used to assess the model performance by comparing the output of the classifier with the real output.

An important premise is that the training and test sets must be completely independent from each other and there are a couple of reasons for that. The first one is known as data snooping bias. Human brains are very good at recognizing patterns, if we tried to gain insights with all the dataset, we would probably select or discard certain models influenced by the patterns we have recognized but this could lead to models that only work well for the particular data we are working with.

The second reason is that if we allow the different models to see all the dataset, they will be overfitted. Overfitting a model means that it will probably have an outstanding performance for the dataset it has learnt from, but it will fail to make correct predictions with other data as the model has been heavily adjusted only for a particular information [12].

We have used two different techniques to divide the dataset: a train test split and a stratified sampling.

1.3.1 Train test split

A train test split is the simplest way of doing the partition. We only have to select a random seed; the seed will select some instances and the instances assigned to the test set will always be the same. In this way, we will have two independent datasets to work as shown in Fig. 1.6.

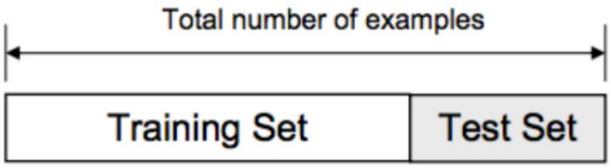


Fig. 1.6 Example of a train test split.

1.3.2 Stratified sampling

Stratified sampling is very similar to the train test split but it changes slightly the approach. By doing a purely random selection we may be introducing certain bias for small datasets. To understand why, we can use the example in Fig. 1.7. If we have to select a group of people to make a survey and we randomly choose the participants, we may end up with an imbalanced dataset as we could have chosen three blue individuals and a green one. To correct this, a stratified sampling strategy can be used as it preserves the proportions of the features in the dataset and the model can learn from a representative sample.

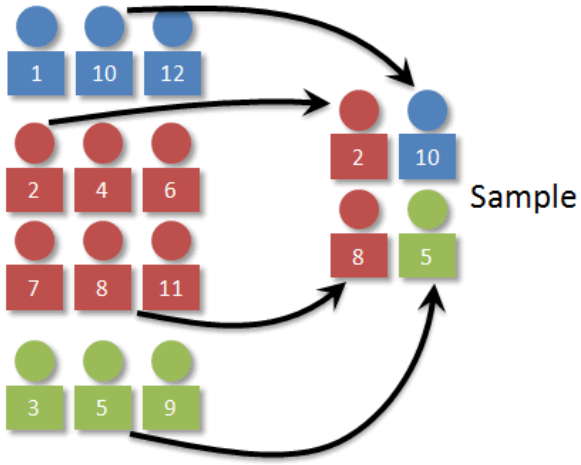


Fig. 1.7 Example of stratified sampling with people **[21]**.

1.3.3 Scaling the data

ML algorithms tend to suffer when the numerical data has very different scales. Some features may have normal values of thousands while others may be under the unit. Thus, it is very important to scale the data. The procedure consists on fitting the scaler on the training set and applying the same scaler to the test set [22]. It is important to do it in that particular order, if we fit the scaler for all the

dataset, we will be cheating as some information from the test set will be filtered into the training set. Furthermore, some models such as SVC assume that all features are centred around zero and have a unit variance so it would be impossible to run them without scaling.

Two different scalers have been used: a MinMax scaler [23] for the Naive Bayes classifier and a standard scaler for the rest of models [24].

1.3.3.1 Standard scaler

The standard scaler process is the following: first it subtracts the mean value (so standardized values always have a zero mean) and then it divides by the variance of all the features so that the resulting distribution has unit variance [12]. Fig. 1.8 shows how the data distribution changes thanks to a standard scaler. On the left-hand side, we can see the data before any treatment whereas the right-hand side shows how the data is distributed when it has been scaled.

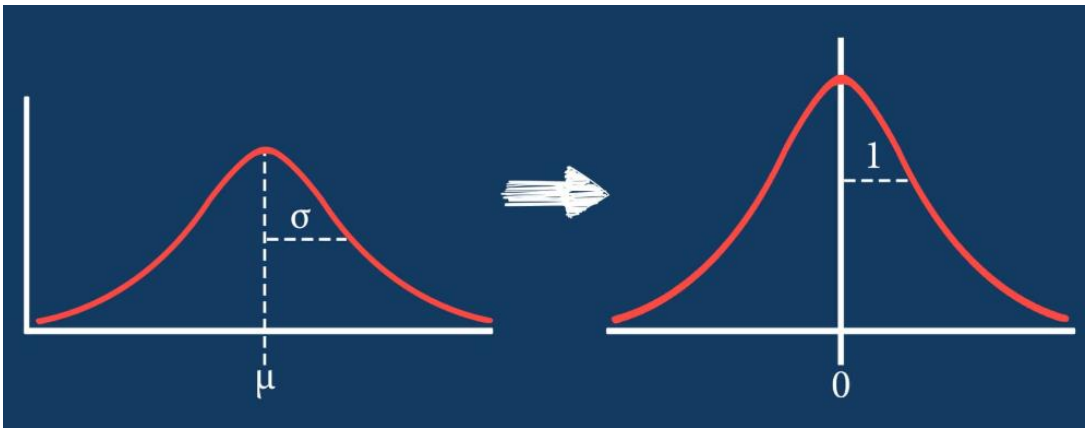


Fig. 1.8 Data distribution before and after a standard scaler **[25]**.

1.3.3.2 MinMax scaler

MinMax scalers are simpler than standard scalers. The only thing they do is to shift and rescale the data so that it ends up ranging from 0 to 1. Anyway, this scaler is far more sensitive to outliers. If there is any mistake in the data and the greatest value is for example twice bigger than the second one, all the dataset will be compressed in a range from zero to one half, and a single point will exist at one [12].

Fig. 1.9 shows the scheme of a MinMax scaler.

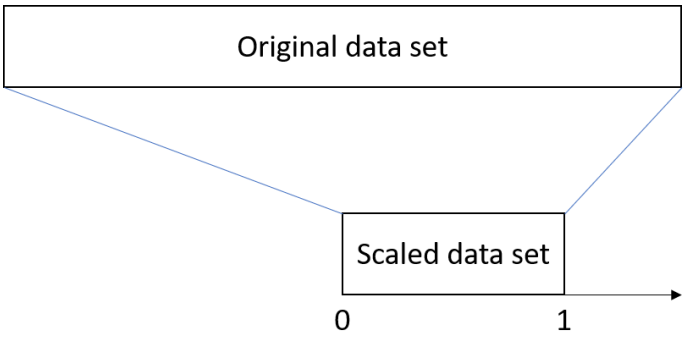


Fig. 1.9 Scheme of a MinMax scaler.

Naive Bayes classifiers, require this data format because they do not admit negative values as inputs.

1.4 Evaluation techniques

To evaluate models, we could just use a couple of sklearn functions called fit and predict with the training and the test set respectively. The first one trains the model and the second one, of course, predicts the results. The problem of doing this comes from the fact that we could end up overfitting our model for the test set. If we try to improve the model performance by tweaking the hyperparameters and we only evaluate it on the test set, we will eventually improve the results of the model but this will happen because some information from the test set is being leaked to the training set.

The correct approach is to train and evaluate the model only on the training set and finally using the test set to assess how well the model does with new or unseen data. Fortunately, there are two sklearn functions called cross_validate and cross_val_predict that help us with this issue.

1.4.1 Cross validation

To understand what cross validation (CV) is, we can look at Fig. 1.10 which shows how the data is managed for this strategy. A fold is just a part of the training set that has been divided using one of the previously mentioned sampling strategies. We need to define a number of folds, which is usually referred as k and the function will apply the following procedure:

- A model is trained using k-1 folds as training data.

- The resulting model is validated on the remaining part of the data (i.e., it is used as a test set to compute a performance measure such as accuracy).

- Finally, the model is evaluated on the original test set.

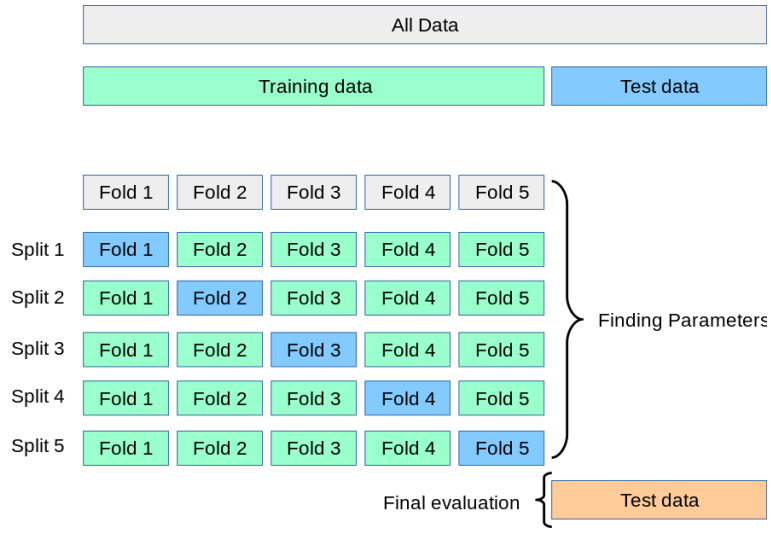


Fig. 1.10 Five folds CV scheme **[26]**.

1.5 Hyperparameters tuning

All models in sklearn have a list of hyperparameters used to control the different aspects of the decision process. They can be divided in two big groups: hyperparameters to control how the solution is obtained and hyperparameters to control when to stop finding a solution.

Regarding the first group, some examples are: the used number of estimators in a random forest (number of decision trees), the solver in an MLP (how to compute the weights of the node connections) or the number of neighbors in a K-nearest neighbors classifier.

Two examples of the second group are: the tolerance in a logistic regression classifier (if the solution does not improve more than the tolerance, it stops looking for a solution) and the maximum number of iterations in a MLP (if the maximum number of iterations is reached, the model stops working even if the solution has not converged yet).

Every hyperparameter has a default value but almost all of them can be changed. This is crucial because certain values that may work perfectly for a given dataset, might be awful for another one. Even though, fine tuning the hyperparameters is a very resource intensive process which can easily last more than one day for

any normal PC. Consequently, unless we have very powerful machines at our disposal, we have to limit it to the most promising models.

In this project, two different tools to explore hyperparameters combinations have been used: grid search and randomized search.

1.5.1 Grid search

A grid search is in fact a very simple technique that relies on gross force. To use it, we only have to define the model that we want to optimize and a parameter grid. A parameter grid is a list with different hyperparameters and the values that we want to try. The grid search function will automatically execute the model as many times as hyperparameter combinations exist and spit out the best solution.

Additionally, we can select how many central processing unit (CPU) cores we want to use for the job. The function to implement a grid search in sklearn is called GridSearchCV and it can be found in the model selection module [27].

1.5.2 Randomized search

Randomized search is very similar to a grid search but instead of trying all the possible combinations, it evaluates a given number of random combinations by selecting a random value for each hyperparameter at every iteration [12]. This method represents two important benefits:

- If we run n iterations of a randomized search, it will explore n different values for every hyperparameter instead of a few values as the grid search strategy.

- The time and resources that the process will consume are more easily controlled.

Sklearn has an available function called RandomizedSearchCV in the same module as the GridSearchCV [28].

1.6 Enhancement techniques

After fine tuning the different ML models, there are certain techniques that may help us in taking one final step in the improvement of the estimators. In this project, we have used voting classifiers, boosters and oversampling.

1.6.1 Voting classifiers

Voting classifiers are not really classifiers themselves but a combination of different ones (ensemble). Their working principle consists on adding the predictions of different ML models according to a confidence score. That means, that the different classifiers will not only emit their predictions but also the probability of that prediction being correct. Thus, the classifiers that are more confident about their guesses will have more weight onto the final decision which is only the sum of all the individual predictions [29].

1.6.2 AdaBoost classifier

An AdaBoostClassifier is also an ensemble method but in this case, we only use one model, typically a decision tree classifier. The algorithm learns by building an estimator from the training data and then creating a second estimator that attempts to correct the errors from the first one. Estimators are added until the training set is predicted perfectly or a maximum number of estimators are added [30].

This classifier is normally used only for binary classification problems so it is likely that it will not improve the results. Anyway, it is always a good practice to try different options and see if something works.

1.6.3 Oversampling

Oversampling is a technique used to adjust how classes are distributed in the dataset. We have particularly used a synthetic minority over-sampling technique (SMOTE). In the five categories problem, the extreme cases will be very underrepresented compared to the other categories and it will be probably very difficult for the models to consider those flights. By using SMOTE, we will create similar instances in the dataset combining data from different flights so that more of them appear and the different models can account for them.

1.7 Performance measures

Performance measures are the tool to assess how good models are with our data. We need to define a common set of parameters so that we can compare the different model’s performance under the same conditions. In this section, the different measures are compared and their relevance and limitations are studied.

1.7.1 Accuracy

Accuracy in classification problems is the number of correct predictions made by the model over all the predictions made and it is expressed as a percentage [31]. Although it is useful, when working with skewed datasets like the one in this project, its use is just indicative. To understand why, we can think of the following classical example: we have a dataset with handwritten numbers from 0 to 9 and we want to make a classifier to know which numbers are not a 5. If the classifier always votes that the image is not a 5, it will have an accuracy of 90% for any well-balanced dataset. Although the result is impressive it is not giving an actual sense of how the model is doing. A much better performance measure can be obtained thanks to the confusion matrix.

1.7.2 Confusion matrix

A confusion matrix is just a manner to visualize how a ML model has classified the instances. To better understand what a confusion matrix is, we will use the following example: if this project instead of predicting 3 different classes, only tried to predict whether a flight landing time is going to be as planned or not a confusion matrix like the one in Fig. 1.11 would appear, where the meaning of the different acronyms is:

- True positives (TP): true positives are the flights with a planned landing time that were correctly predicted as planned.
- False positives (FP): false positives are the flights with a non-planned landing time that were wrongly predicted as planned.
- True negatives (TN): true negatives are the flights with a non-planned landing time that were correctly predicted as non-planned.
- False negatives (FN): false negatives are the flights with a planned landing time that were wrongly predicted as non-planned.

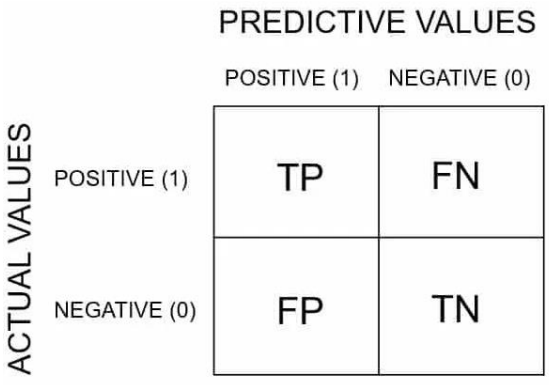


Fig. 1.11 Example of a 2x2 confusion matrix.

From the above confusion matrix, three new performance measures are obtained: precision, recall and f1. Although they are often confused with accuracy, they do not represent the same measure [31].

1.7.2.1 Precision

Precision tries to answer the question: which is the correct proportion from all the positive predicted values? In our example that would be: from all the instances that have been predicted as planned, how many of them are really planned? To calculate the response, we must use equation 1.1.

$$Precision = \frac{TP}{TP + FP} \quad \textbf{(1.1)}$$

A precision of an 80% means that from all the instances that were predicted as planned by the model, 8 out of 10 were correctly predicted.

1.7.2.2 Recall

Recall, on the other hand, tries to answer what is the proportion of real positives that were correctly identified? Getting back to our example, that would be, from all the instances that are actually planned, how many of them have been correctly classified? In this case, we have to use equation 1.2.

$$Recall = \frac{TP}{TP + FN} \quad \textbf{(1.2)}$$

Now, a recall of an 80%, means that from all the planned instances, 8 out of 10 have been correctly classified.

1.7.2.3 f1 score

The f1 score combines the precision and the recall into a single metric which is very convenient because it can be used to compare different classifiers. It is defined as the harmonic mean of precision and recall and it is calculated with equation 1.3.

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 * \frac{precision * recall}{precision + recall} = \frac{TP}{TP + \frac{FN + FP}{2}} \quad \textbf{(1.3)}$$

The harmonic mean is used instead of the regular mean because in that way the f1 value is only good when both precision and recall have also good values.

1.7.2.4 Precision/recall trade-off

Unfortunately, there is a trade-off between recall and precision, this means that we cannot have an excellent precision and recall at the same time. Fig. 1.12 shows a precision-recall curve for our three classes problem under a LinearSVC model. The blue, turquoise and orange lines represent the behaviour of the advanced, planned and delayed categories respectively when analysed individually (the precision and recall are obtained for each category from the confusion matrix). The yellow line, shows the behaviour of all the categories at the same time. Additionally, the plot contains iso-f1 curves to observe what f1 values our model is achieving.

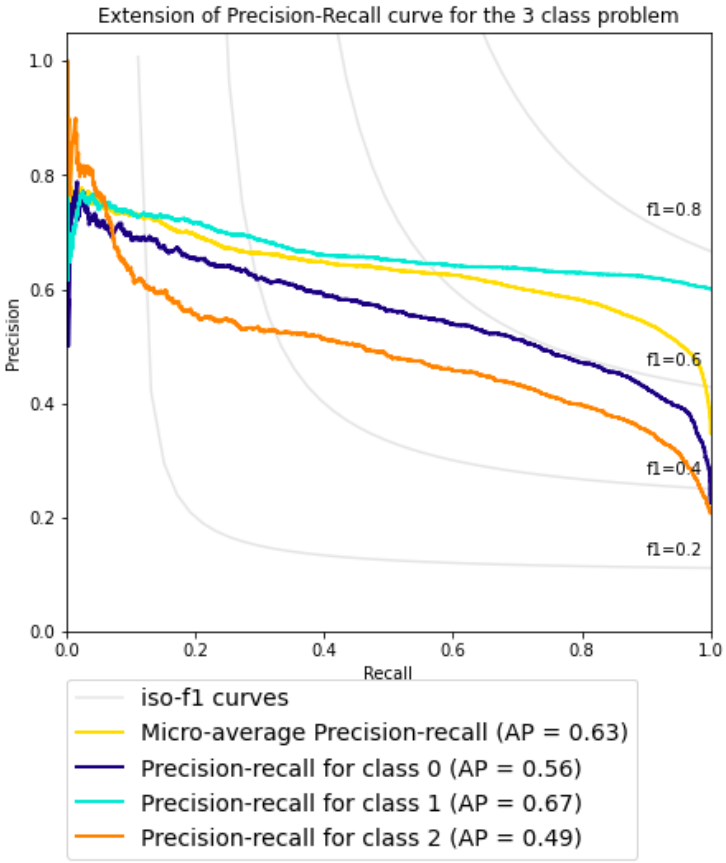
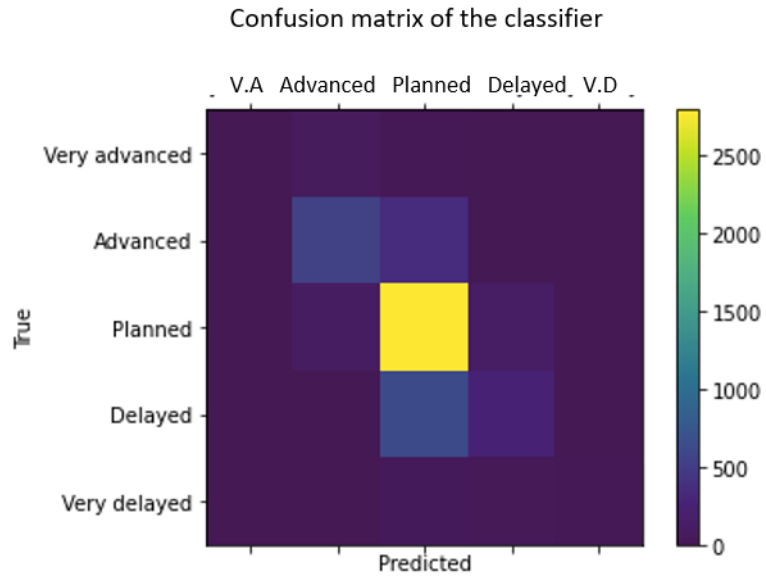


Fig. 1.12 Precision/Recall curve for the three classes problem.

Very interesting conclusions can be drawn from the previous figure: we can obtain perfect recall or perfect precision; the planned category allows us to improve the recall without scarifying much precision as the curve is more or less flat but we cannot do that with the advanced and delayed categories; the optimum f1 value is well over 0.6 (0.68 in fact).

The decision of aiming at a high precision, a high recall or an intermediate solution depends on the problem. If we use a ML model to recognise thieves with video cameras, we do not want to miss any thieve (high recall) although we will stop innocent people (low precision). In this project, we are not particularly interested on a high recall or a high precision so we will try to optimize the f1 and the accuracy.

To generalize the concepts, Fig. 1.13 shows a real five class confusion matrix. The elements on the main diagonal are the correctly classified instances. As an example, the third row and fourth column element, indicates that 130 planned instances were wrongly predicted as delayed. Also, it can be seen that the confusion matrix has a colour legend that associates colours with the number of instances. Thus, at a first glance, we can observe a yellow tone in the middle of the matrix meaning that more than 2,000 planned instances were correctly classified.




```
Confusion matrix: [[    7    76    19     0     0]
                   [    3   556   355     0     0]
                   [    0   108  2794   130     0]
                   [    0     2   631   274     4]
                   [    0     1    60    27    12]]
```

Fig. 1.13 Example of a five classes confusion matrix.

In this case, to calculate precision and recall, we need to add up all the elements of the class. The recall of the planned category is computed in 1.4.

$$Recall = \frac{TP}{TP + FN} = \frac{2794}{108 + 2794 + 130} = 0.9215 \quad \textbf{(1.4)}$$

CHAPTER 2. DATA COLLECTION AND PREPARATION

This chapter explains how we can go from a theoretical problem such as predicting the time to land of an aircraft to the point in which a ML model is ready to start making predictions. The final output is the matrix that is used as an input for ML and in this case every instance corresponds with a flight arriving to Barcelona and every feature is one of the characteristics of this flight listed in Table 2.1.

Table 2.1 Features of the ML input matrix

Feature category	Feature	Feature type
Position	Latitude	Float
	Longitude	Float
	Altitude	Integer
	Heading	Float
Speed	Indicated airspeed (IAS)	Integer
	Ground speed (GS)	Integer
Meteorology	Wind direction	Float
	Wind speed	Float
	Wind variability	Float
	Visibility	Integer
	Barometric pressure	Integer
Airport	Runaway	String
	Airport azimuth	Float
	Airport distance	Float
Traffic characteristics	Landing category	String
	Mix index	Float
	Type of airline	Integer
Time	Day of the week	Integer
	Initial time	Integer

The whole process, described in this chapter, consists of four basic steps:

1) Deciding the limits of the project, the conditions that we will focus on and what are the desired outputs.

2) Obtaining the most relevant data for the project and converting it into an adequate format.

3) Preparing the data according to the criteria established in point one.

4) Generating a matrix that can be understood by a ML model so that it can start making predictions with it.

This is clearly summarized in the scheme of Fig. 2.1.

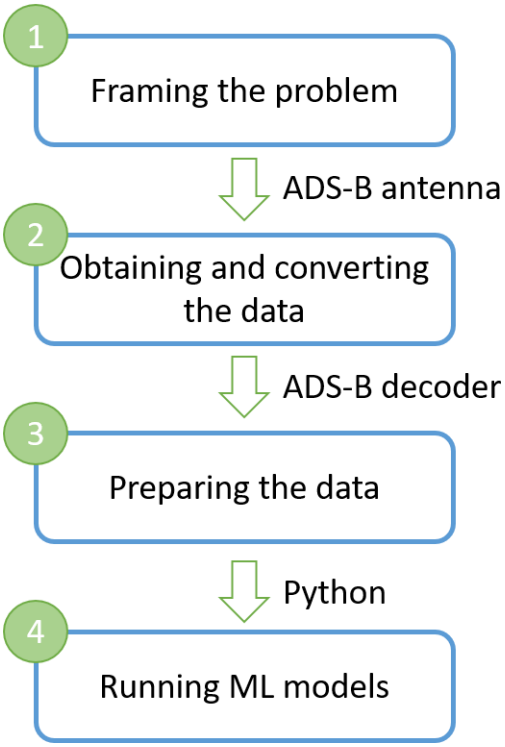


Fig. 2.1 Initial scheme of the project development.

2.1 Scope of the problem

2.1.1 Desired outputs

As we said before, we will first try to predict three different classes. To define them, we need to prepare all the data and finally learn what most planes do. If most of them take three thousand seconds, that will be the planned time to land and with that reference, we will be able to define what advanced or delayed means.

The same thing has to be done with the very advanced and very delayed categories although their landing time, will represent the rarest scenarios.

2.1.2 Area of interest

To predict how fast or how slow will aircraft land in a certain area, we need to determine what is the most interesting one. As the project is focused in

Barcelona, there are three possibilities: Barcelona's flight information region (FIR), standard arrival routes (STAR) or approaches. During most parts of the en-route or descent phase, there are not substantial changes in the speed or the trajectory (i.e. planes fly a given path at a constant speed). On the other hand, before entering the approach, controllers often instruct aircraft to hold, to change their heading or to stay below a certain speed creating a complex environment in which it is difficult to predict what planes will do.

Considering the speed and trajectory variability, the most interesting area is the one comprising the end of the STARs and all the approaches until the intermediate fix (IF) as it is the most diverse one.

The IF is not considered because, as it can be seen in Fig. 2.2, when aircraft arrive at that point, they have to follow the instrumental landing system (ILS) and no significant changes can occur.

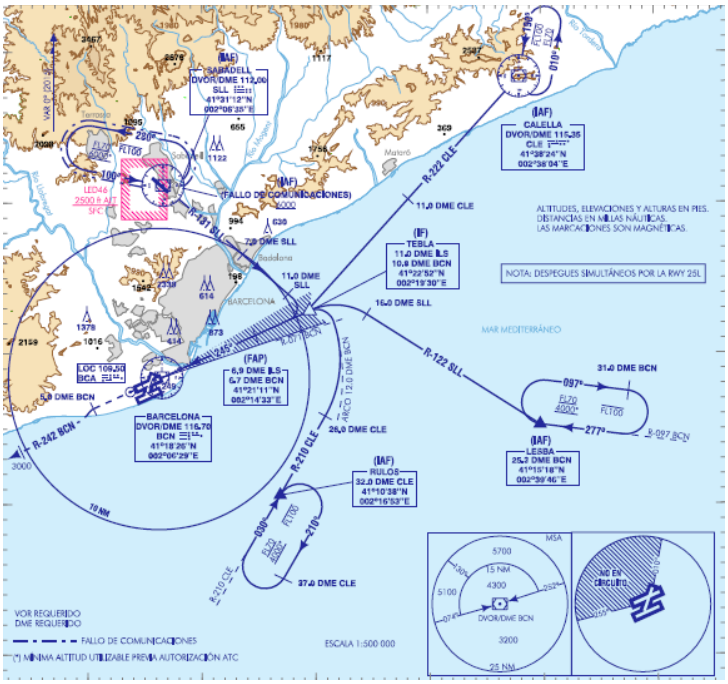


Fig. 2.2 Instrumental approach chart to runaway 25R.

2.1.3 Problem boundaries

Defining the problem boundaries consists on transforming the area of interest into something tangible. If we want to make predictions, that means that we shall not wait to know all the variables until the plane landing. We have to define a point at which predictions must be made.

To select the best boundary, we also looked at the Spanish aeronautical information publication (AIP). As it can be seen in Fig. 2.3 there is a control area

(CTA) with a pink circular shape centred at the airport. This kind of airspaces are very interesting since a lot of air traffic controllers (ATC) action is required.

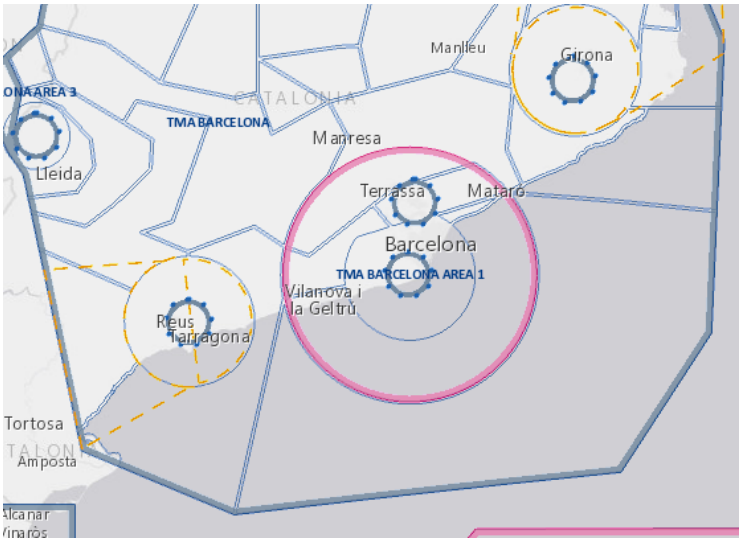


Fig. 2.3 CTA in Barcelona (INSIGNIA map).

Combining the area of interest with the airspace on top of the airport, we determined that the most suitable boundary would be a 68 Km circle with its centre in the aerodrome reference point (ARP). This circle comprises all the area of interest and it exceeds by 20 Km the limits of the CTA. The result with the 25,294 flights in the project can be seen in Fig. 2.4.

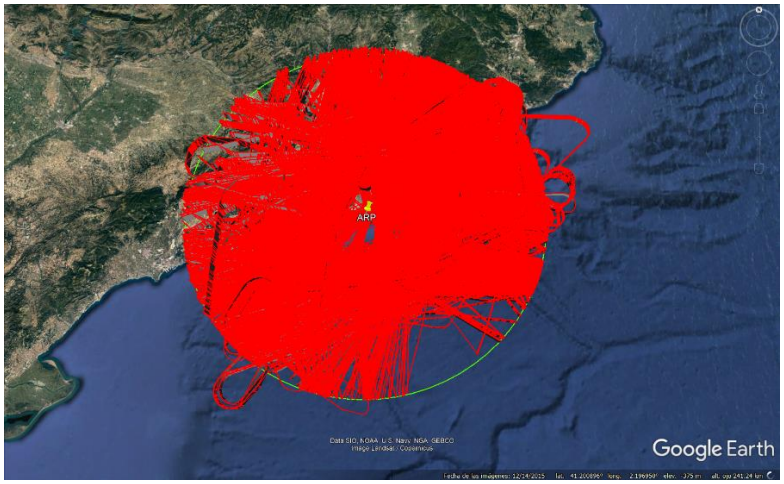


Fig. 2.4 Final boundary with flights.

An example on the actual implementation, is an arriving flight as an instance of the input matrix and all the features of the flight obtained at the moment in which the aircraft enters the circle.

2.2 Data collection

2.2.1 ADS-B

Automatic data surveillance broadcast (ADS-B) is a data-link broadcast mode in which aircraft share their identification, position, sensor outputs and any other relevant information to ground stations or other aircraft [32]. To obtain the raw data, the easiest way is to analyse ADS-B reports using tools to decode messages that are already available [33].

To obtain these reports, an ADS-B antenna located on EETAC's rooftop was used. This antenna receives signals from nearby aircraft, uploads the messages to a server called radarcape where a python script can download the raw data. Additionally, all the received information is shared with a website called: www.flightradar24.com which provides in return a free premium account. The antenna setup can be seen in Fig. 2.5.

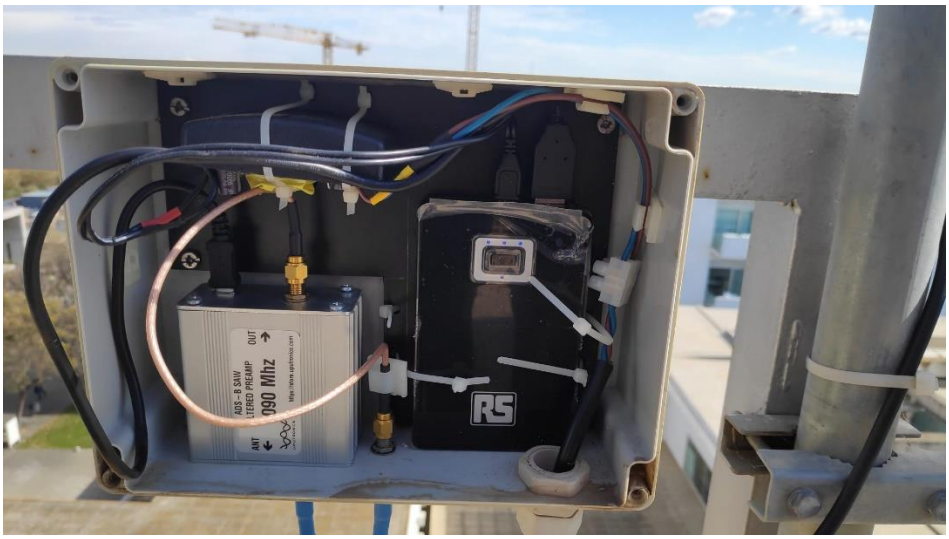


Fig. 2.5 Antenna setup on EETAC's rooftop.

It is worth mentioning that ADS-B reports suit perfectly the objective of this project as they contain an enormous quantity of information (many features or columns) and more than 850 planes fly into Barcelona each day [34] (many instances or rows). Also, the antenna has enough range to capture all the planes within the area of interest. In fact, some of them can be tracked until the south of France, or southern regions in Spain.

2.2.2 ADS-B Decoder

The available ADS-B reports were stored in a server and they were in a DAT format. These files do not present the information in a “human” way as they are written in codified binary format and the information is sorted by time. Whenever the antenna captures a signal, it writes the data along with a time stamp but it does not any further treatment. To process it, we need to use an ADS-B decoder.

All aircraft around the world are assigned what is called an international civil aviation organization (ICAO) hexadecimal code. This code is composed of numbers ranging from 0 to 9 and letters from A to F and its purpose is to identify each aircraft in the world in a unique manner [35]. The ADS-B decoder that we used is written in C# and it sorts all the information, so that each aircraft (each hexadecimal code) has all its information arranged in a chronological manner. Anyway, there is a slight inconvenient on ordering the information in this way which is further discussed in 2.3.1.1.

It is also worth mentioning that the ADS-B decoder creates different lists for the different features of the flights but their time stamps do not match. If we think about the coordinates and the speed for example, we may think that both the list of coordinates and the list of speeds have the same length and each position has the same time stamp but this is not true. Aircraft transmit their information at different moments and with different frequencies.

2.2.2.1 Direct and indirect features

The features that we used can be divided in direct and indirect. The term direct makes reference to the fact that these features can be directly obtained from ADS-B reports but there are many others which are not. A good example is what happens with the runaways. It is evident that knowing the runaway in which a plane is landing can be a valuable information for a ML model, but this information cannot be obtained from the ADS-B decoder because aircraft do not transmit it. To obtain these other features, that we have defined as indirect, we have two possibilities: some of them can be obtained from the direct ones such as the runways (information on how to obtain them can be found in 2.3.2.2) and some others have to be obtained from external sources such as the meteorological terminal area reports (METARS).

2.2.3 From ADS-B decoder to data preparation

The ADS-B decoder allowed us to gather the direct features but much more development was still needed. The indirect features had to be obtained, the data had to be prepared according to the previously defined boundaries and the output matrix had to be generated. We modified the existing ADS-B decoder to dump all the data into a temporary file which could later be imported by a python script to continue the process.

To sum up, Fig. 2.6 shows the data collection process that we followed: we used an ADS-B antenna to obtain reports, we de-codified these reports with a modified ADS-B decoder and we sent the data to python to continue with the development.

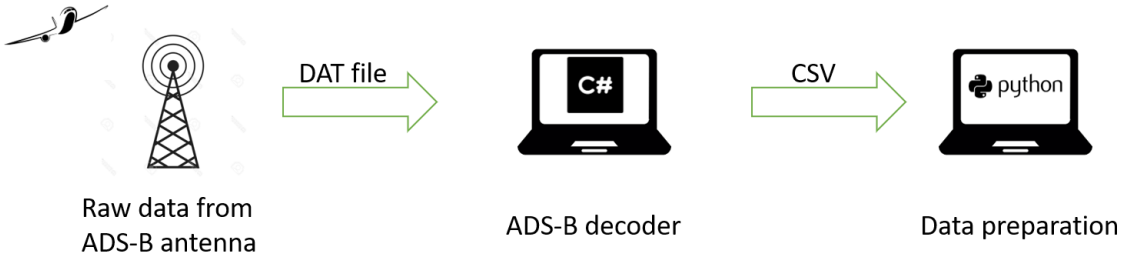


Fig. 2.6 Data collection process.

2.3 Data preparation

The data preparation program is responsible of generating useful functions that can convert the type of data. An example is a function to go from a position given in degrees, minutes and seconds as a string into a decimal degree float. Also, the are certain libraries that automatically calculate the distance between two given points on earth that can be very helpful. Some important coordinates must be defined such as the ARP or the touchdown zones for the five runaways in the airport.

There is also a module called PICKLE that stores all the objects that we will use in a very efficient manner, so that they can be “unpickled” or recovered without executing all the code again, hence saving a lot of time [36]. Additionally, we created a class called “Instance” and every time a new flight was prepared a new class object was generated and the information was assigned to the different attributes. The attributes are of course the different features that we need to prepare.

Appendix A contains a scheme and a brief explanation on how the code is structured.

2.3.1 Direct features preparation

2.3.1.1 Coordinates and altitude

The coordinates and the altitude are probably two of the most important features for the ML models. A high altitude when crossing the circle, could mean that

planes are not going straight forward to the airport, on the other hand low altitudes could mean that they will be landing soon. A similar thing happens with the coordinates. If an aircraft enters the circle and it is already flying an approach this means that it will be landing fast but if the plane is entering through a “weird” point this could mean that it has been vectorized or that it is suffering some kind of delay.

In the ADS-B decoder section it was said that the way in which the program structures the information creates a little problem. If we imagine an aircraft based in Barcelona with four scheduled flights, it may start its day and go to Mallorca, then come back to Barcelona, go to Paris and then come back again. This is logical and this is what happens with the majority of planes but the problem is that the coordinates of all the flights are stored in the same list because they belong to the same ICAO hexadecimal code. The graphical representation of the issue can be seen in Fig. 2.7 and a real vertical profile of an aircraft can be seen in Fig. 2.8.

Fig. 2.7 Single list of coordinates problem.

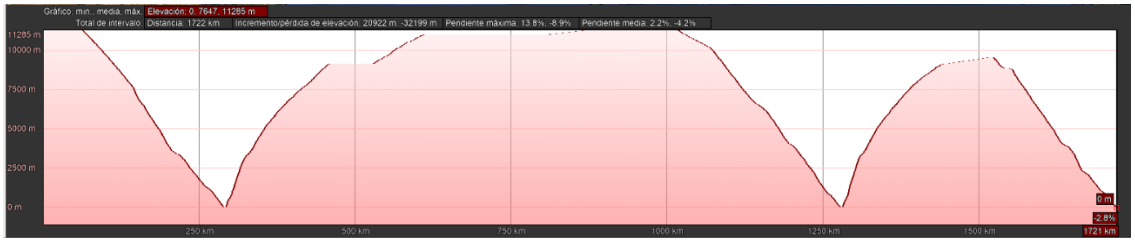


Fig. 2.8 Vertical profile of a real aircraft.

We are only interested on the arrivals as our goal is to predict what will be their landing time. Keeping up with the previous example, we need two different lists, one for the flight arriving from Mallorca and another one for the flight arriving from Paris. To solve this issue the following criterion can be applied:

1) If two consecutive coordinates are separated more than 600 seconds (10 min) they belong to different flights. This will happen between the last coordinate of an arrival and the first coordinate of a departure.

2) For a flight to be considered as an arrival, the altitude of the first coordinate should be at or above 350 meters and the last one should be at or below 25 meters.

3) For the flight to be considered as valid, any coordinates should be separated less than 30 Km. As it can be seen in Fig. 2.9 some flights do not fulfil this condition since errors occurred during the transmission of the message.

Fig. 2.9 Erroneous data from the ADS-B antenna.

Once all the arrival lists are correctly separated, the coordinates of the first point immediately after the 68 Km circle are extracted and assigned to its corresponding attribute in the instance class and the same thing is done with the altitude.

2.3.1.2 Heading

The heading importance, comes from the fact that it shows where the nose of the plane is pointing to. When ATC are sequencing arrivals, it is a very common practice for them to give headings as an instruction. ML models will perhaps detect what are the most common headings under normal conditions and what are the headings when ATC are involved.

The only thing that has to be done is calling a function that can calculate the azimuth between the first two points of the arrival lists and then assigning it to its corresponding attribute.

2.3.1.3 IAS and GS

IAS is a capital speed in aviation since all the ATC speed indications are expected to be referenced to that magnitude. In the same way as with heading, a very common indication from an ATC could be to stay below a certain speed. Thus, IAS gives an idea of how fast a plane is flying. Although IAS is useful, as we are trying to predict the time planes will take to land, GS is even better. This speed is in fact used to calculate the flight time, as it is the real speed the plane has over the ground. If the IAS is fast but there is a strong headwind, it will result in a slow GS and a slow landing.

To obtain the IAS and the GS from ADS-B reports a little problem appears. As we said before, the time stamps of the different features do not match and the only solution is to look for the closest value (with respect to the coordinates) and assume that a certain error is being introduced.

It is also worth mentioning that in some cases the IAS, the GS or both of them present erroneous values such as extremely low speeds or no available information. To correct it, the following guidelines are followed:

1) If the IAS is incorrect or not available, the value of IAS is set to 250 Knots (kt) and the value of the GS remains the same.

2) If the GS is incorrect or not available, the values of GS is set to be as the IAS and the IAS remains the same.

3) If both the GS and the IAS are incorrect a value of 250 kt is assigned to each one.

2.3.1.4 Initial time

The initial time affects the speed of planes in two different manners. The first and most evident one is that the number of flights in any aerial network is strongly dependent on time. Fig. 2.10 shows the opening scheme of the different sectors of the terminal manoeuvring area (TMA) of Barcelona. As it can be seen, during nights, from midnight until 4 AM there is only one sector available as the airport is usually closed. On the contrary, when looking at the peak hours of the airport, the standard configuration is working with seven sectors at the same time. The number of sectors directly influences how ATC will sequence aircraft and this may be detected by ML models.

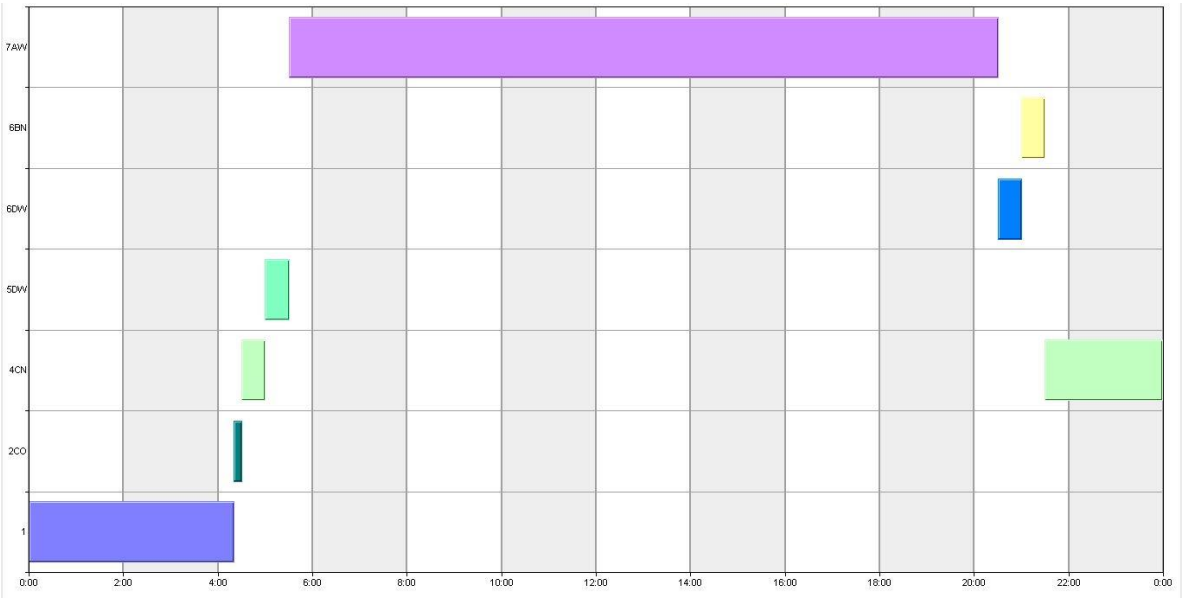


Fig. 2.10 Opening scheme in Barcelona ACC.

The second reason is that as the day goes by, the probability of suffering delays increases as every flight is affected by possible incidences during the preceding hours and this can lead to unexpected overloads of the network.

To obtain the initial time, we only need to check the first coordinate that we selected (the one crossing the circle) and extract its timestamp.

2.3.2 Indirect features obtention and preparation

This section explains how the indirect features are obtained, prepared and their overall importance. It is important to remember that these features are not less relevant than the direct ones but they cannot be directly obtained from ADS-B reports because the information is not transmitted.

2.3.2.1 ARP distance and azimuth

The ARP distance should not have a strong influence since the 68 Km circle from the azimuth has already been set as a common start for all flights. Even though we can use it as a double check. If the assumption that it will not have a strong influence is right, ML models should not be influenced by this variable. On the other hand, sometimes aircraft do not transmit their position for a long time, that means that the first coordinate instead of being at 68 Km could be at 60 Km. In this way the different models can account for this difference only for those particular cases.

Regarding the ARP azimuth, it has a crucial importance. Whereas heading measures the angle difference between the nose of the plane and the magnetic north, the ARP azimuth measures the angle between the nose of the plane and the ARP of the airport. Thus, it indicates where planes are coming from when approaching to the airport. In combination with the heading, very interesting conclusions can be obtained. For instance, an aircraft could be entering the circle from the north but its nose could be pointing to the east, suggesting that it may not go directly to the airport. On the contrary if the plane was entering through the east and the heading was also pointing eastwards, this could mean a rapid approach to the airport. The difference can be appreciated in Fig. 2.11 where the blue line shows the heading and the orange one the ARP azimuth.

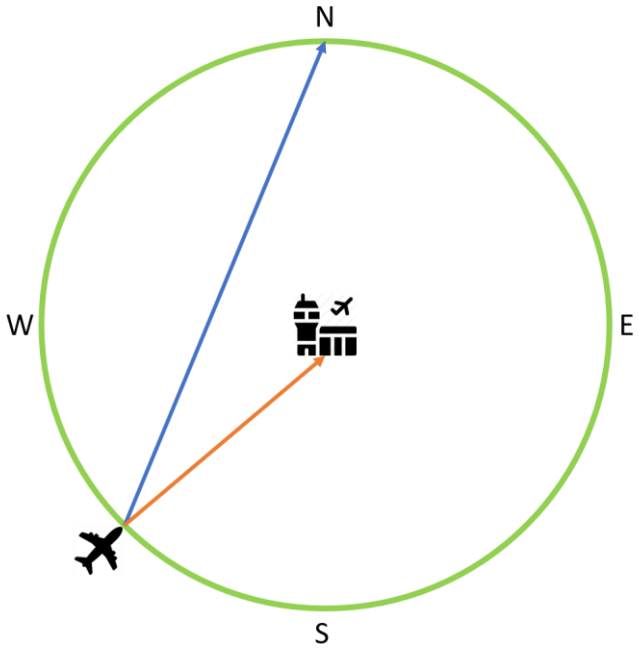


Fig. 2.11 Difference between ARP azimuth (orange) and heading (blue).

Two obtain the ARP distance and azimuth we only need to use the same library as with the heading. We select the entering point of the circle and the previously defined ARP coordinates and it will automatically compute the desired parameters.

2.3.2.2 Runaways

Knowing which runaway is in use for landings is important. Depending on the active runaways at an airport, aircraft will fly different procedures and approaches that directly influence the speed they will have. Barcelona airport has three runaways and the typical configuration is 25R for arrivals and 25L for departures,

or 7L and 7R in case the wind direction is changed. During nights aircraft typically land at 02 and 20 is never used [37].

To obtain the runaways, we used a method which consists on defining five circular touch down zones which can be seen in Fig. 2.12. Later, we go over the arrivals list and we check which coordinates fall inside the red zones.

Fig. 2.12 Touch down zones at Barcelona.

By using this method to obtain the runaways, we are using coordinates which are beyond the circle (in fact at the very end of the flight). Anyway, this is completely fair since this feature could be easily obtained in the loop by other means. (i.e. runaways can be obtained at the boundary of the 68 Km circle or even before but not with ADS-B reports).

2.3.2.3 Meteorology

The meteorological features that we will use are: wind direction, wind variability, wind speed, barometric pressure and visibility. All these parameters have a common influence, which is reducing or increasing the airport capacity but they do it in different ways. The wind direction has a very strong influence on crosswinds. Large wind speeds along with a high variability can create a phenomenon called wind shear. When this happens, aircraft cannot land and they have to start flying holding patterns which substantially increase the landing time.

Low barometric pressures, indicate the presence of storms whereas high pressures are associated with good days to fly. Finally, if the visibility is dramatically reduced, airports can start the so-called low visibility procedures (LVP). Under these conditions the operations are significantly affected and this can lead to severe delays.

Although ADS-B reports contain some meteorological information, we decided that it was more sensible to use the meteorological conditions at the airport. This is because if we use the data directly read from the aircraft sensors, local phenomena like a strong gust of wind could lead ML models to wrong conclusions when the conditions at the airport could be excellent. To obtain them, METARS were used. A METAR is just a standard format to transmit meteorological information. In the case of Barcelona, they are published every half an hour and they contain all the relevant information that ATC and pilots need to know [38].

METARS were obtained from the webpage: www.ogimet.com. It allows to download large text files containing information from a whole month. Several files can be downloaded and arranged together so that they can be read later with python. An example of how these files look like can be seen in Fig. 2.13.

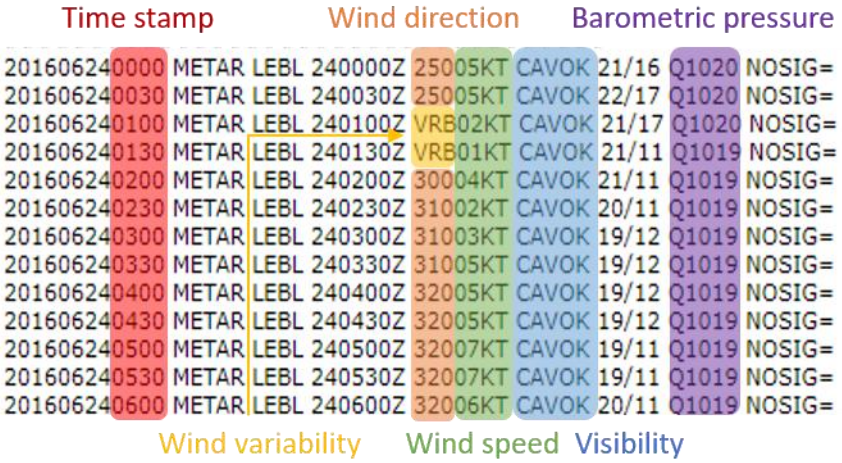


Fig. 2.13 Text file with METARS obtained from OGIMET

Fortunately, to decode METARS, there is a library with the same name created by Tom Pollard which handles almost all the possible cases. All the files can be decoded with the library and only some rare cases that cannot be fully decoded require manual action. Even tough, these weird cases do not have a direct influence on the features that we are searching, thus the problematic parts can just be erased.

As METARS are published every half an hour, the moment in which aircraft enter the circle does not exactly match the conditions at the airport. Anyway, this is not particularly relevant because meteorological conditions do not change extremely fast. We can consider that the 30 minutes gap is short enough to assume that

conditions remain the same during the period. This means that the meteorological information is obtained from the closest 30 minutes gap in which the aircraft crosses the circle boundary.

The visibility, the wind speed and the barometric pressure can be directly assigned without any further treatment but the wind direction and the wind variability require a little bit of processing. The wind direction is just the angle where the wind is coming from (with respect to the North) and the wind variability is a number between 0º and 180º indicating how much does the wind direction vary from the predominant one. There are four possible cases:

1) If both features are correct, they are directly assigned.

2) If the wind direction is correct but there is no information about the variability, the first one is directly assigned and the second one is set to zero.

3) If the wind variability is correct but there is no information about the wind direction, the first one is directly assigned and the second one is set as a random integer between 0º and 359º.

4) In all the other cases, the wind direction is set as a random integer between 0º and 359º, and the variability is set as 180º.

By taking the above criterion, some error is being introduced as some features have random values. Even though, it is preferable mainly for two reasons: some diversity is introduced in the data which could reduce bias for certain cases and some instances that would otherwise be lost can be kept for the study.

2.3.2.4 Landing category

The landing category or wake turbulence category (WTC) is a classification of aircraft depending on their maximum take-off weight (MTOW). There are currently four categories according to ICAO: light, medium, heavy and super [39]. The importance of the landing category comes from the fact that ATC separate aircraft depending on these landing categories as it can be seen in Fig. 2.14.

ICAO WTC scheme		Follower			
		Super (A380-800)	Heavy	Medium	Light
Leader	Super (A380-800)	(*)	6 NM	7 NM	8 NM
	Heavy	(*)	4 NM	5 NM	6 NM
	Medium	(*)	(*)	(*)	5 NM
	Light	(*)	(*)	(*)	(*)

*Indicates that minimum separation is minimum radar separation (MRS).

Fig. 2.14 Aircraft separation depending on landing category.

In Barcelona the vast majority of aircraft fall inside the medium category. This is because the biggest airlines that operate in the airport are low-cost carriers with a typical fleet composed of A320s and B737s. Even though, there are also some heavy aircraft, specially from the airlines that fly to America or Asia, some light aircraft which correspond to general aviation and two daily A380s from the Dubai based company Emirates. Given this variety, this feature may be useful for ML models to make better predictions.

To obtain this feature two steps are needed. The first one, of course, is to obtain the aircraft type. As the ICAO hexadecimal code uniquely identifies every aircraft in the world, we can use free public databases that relate the code with the aircraft type. After that, and thanks to other public databases which relate every single aircraft model in the world with its WTC the feature can be obtained and assigned.

2.3.2.5 Mix index

The mix index is strongly related with the landing category and it is just a measure of how heterogeneous the traffic is. Its importance comes from the fact that although the landing category already offers a measure of the type of traffic for every single aircraft, the mix index helps to understand the interactions between them [40]. The equation to calculate it is 2.1.

$$Mix\ index\ (\%) = C + 3D \qquad \textbf{(2.1)}$$

Where C is the percentage of light and medium aircraft and D is the percentage of heavy and super aircraft. The original equation only considers the medium ones for C and the heavy ones for D but we slightly modified it to include all the different traffic types. A mix index of 100% means that only light or medium aircraft are present, on the other hand, a mix index of a 300% means that only heavy or super aircraft are present. Between these two cases, there are a lot of possible combinations that result in intermediate mix indexes.

We assigned to every instance the mix index of the airspace when it was crossing the circle. That means, that all aircraft that have already crossed the circle but have not landed yet account for the calculation of the parameter. This should give an idea of how complex the situation is upon arrival.

2.3.2.6 Day of the week

Another interesting feature that we selected is the day of the week. This feature should not have an enormous impact but ML models can account for many variables and it will be interesting to see what is its weight in the decision process. It can be directly obtained from the time stamp of the first coordinate.

Fig. 2.15 shows the mean number of flights per day during year 2016 in Europe. It is remarkable to see how weekends are by far the less crowded days in the European network whereas Fridays can allocate up to five thousand more flights in a single day.

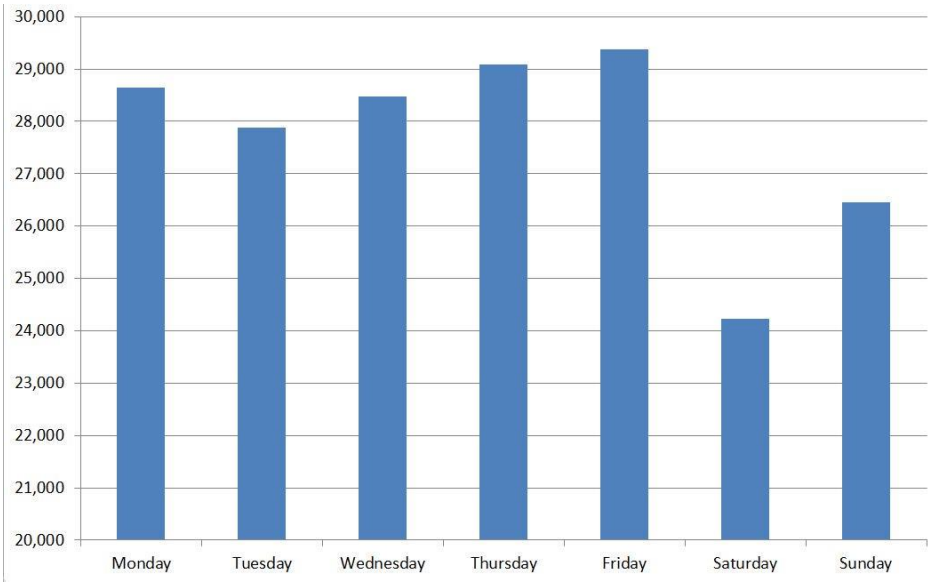


Fig. 2.15 Mean number of flights per day during 2016.

2.3.2.7 Airlines

The airline type, as the day of the week, should not have a strong influence on the landing time because ATC should not be suggested by it. Its purpose is simply to check if that assumption is right and see if any bias is present.

Four different types of airlines were defined: European airlines, American airlines (for USA and Canada), Latin American airlines (for central and south America) and other airlines (mostly composed of Russian and Asian airlines).

Every instance will have four attributes corresponding with the four types of airlines and all of them will be null except for the one containing the carrier, which will be set to one. To obtain the airline type, we can use the callsign since the first three letters indicate the company. After obtaining the letters, we compare them with a custom database that includes all the airlines operating in Barcelona and their classification (European, American, etc.)

2.3.3 Exceptions

The code we designed in python, only needs a temporary file and if it has the correct format, the program will work well. Unfortunately, there are some cases which are very difficult to predict and wrong or misleading values can appear in the different features. Before creating the final matrix that will be used as an input for ML models, some data postprocessing is needed. When screening the different attributes some mistakes were found mostly because of errors in the data.

An example is the obtention of the landing categories for some arrivals. We said that they can be obtained from a public data base and although this is true, in some cases there is not any log for certain hexadecimal codes. Thus, a web scrapper has to be designed to obtain the same feature from webpages such as www.airframes.org. This process is valid but very time consuming since every request will last at least 15 seconds compared with an almost immediate search in a database.

Fig. 2.16 shows another example of incorrect data. The vertical profile of the arrival is correct since it ends at zero meters but when looking closely at the coordinates it can be seen that the plane is supposed to be landing on a taxiway parallel to the runaway.

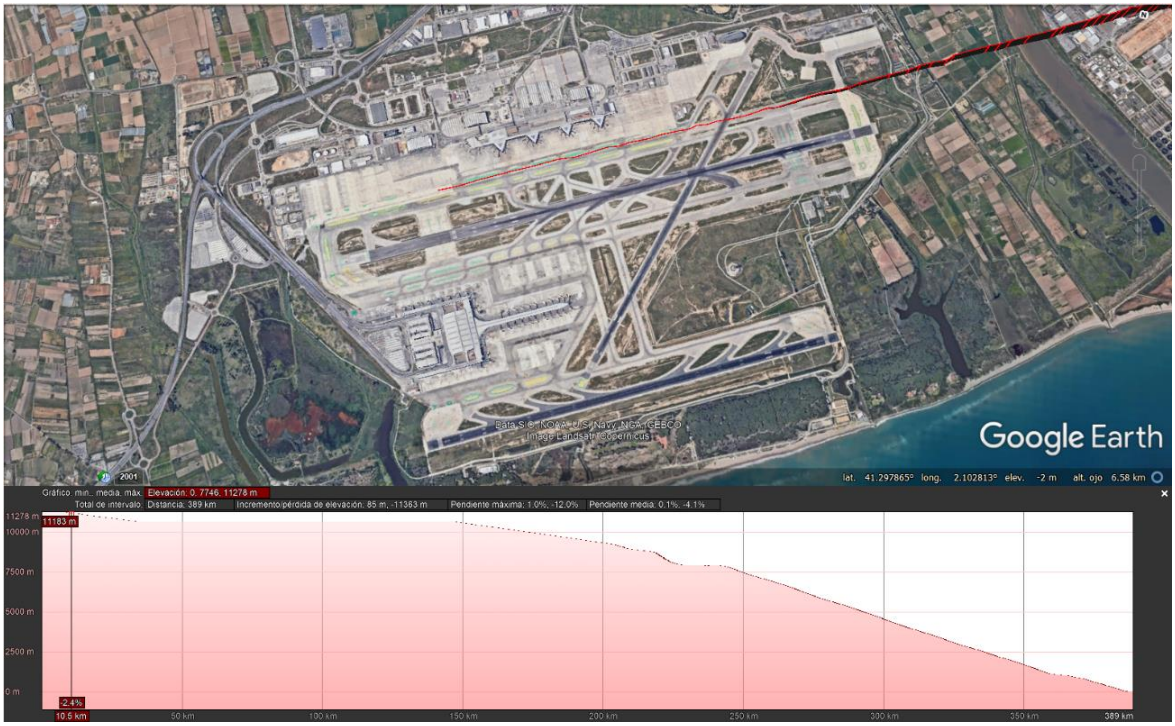


Fig. 2.16 Data error example.

A final example of erroneous data is the case of two planes that according to their coordinates and altitude, were landing near Marseille although it is completely

impossible for the antenna to capture a signal at such a low altitude and so far away.

Finally, an example of a difficult case to predict and handle can be seen in Fig. 2.17. It shows a go around, a manoeuvre which consists on returning to the air when an aircraft is close to the runaway due to technical issues or any factor leading to an unsafe approach. This case is particularly difficult when two or more missed approaches occur.

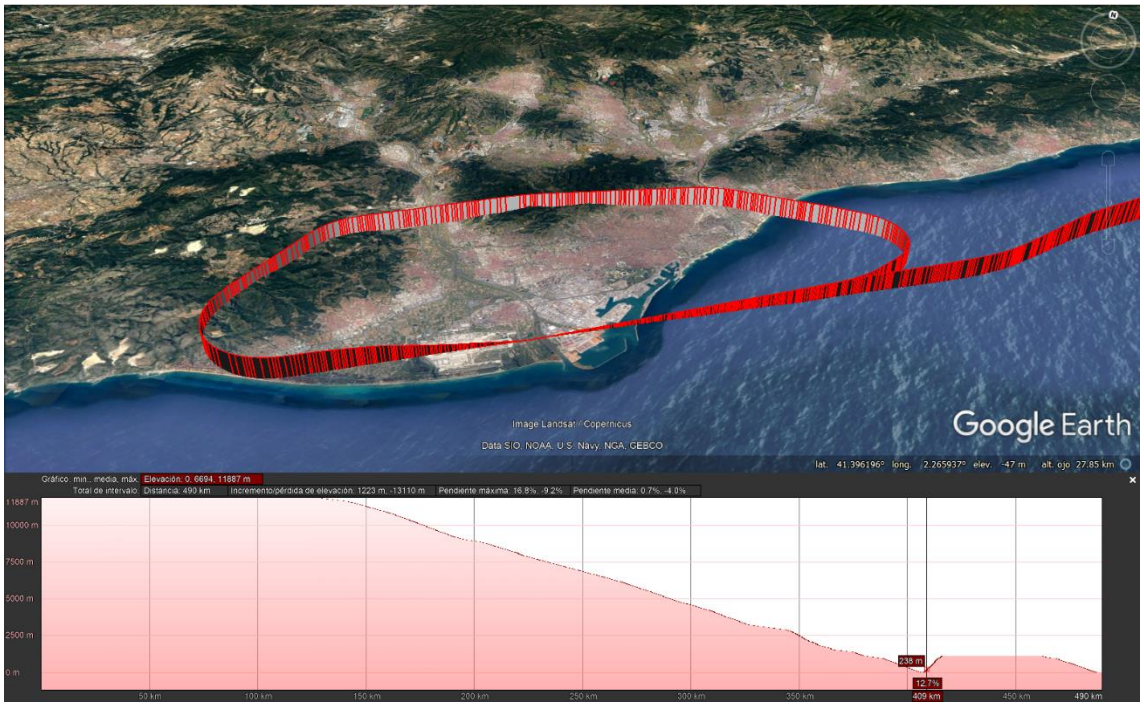


Fig. 2.17 Example of a go around in Barcelona.

2.3.4 Time to land

As we said before, when using ML, all the dataset (the input matrix) is divided in two different parts: a test set and a training set. Regardless of the set, the “solutions” are needed. The training set needs them to learn as it will try to understand why all the features result in a time to land being advanced, delayed or planned. The test set, on the other hand, needs the “solutions” to assess how good it did at predicting the time to land.

Taking that into account, we need to define a final feature or column called time to land. It is simply calculated by subtracting the time in which the aircraft lands to the moment in which it crosses the circle. The process of converting the time to land into different categories is explained in 2.4.1.3.

2.4 Final matrix generation

The final matrix that will be fed to ML models is the objective of all the process described in CHAPTER 2. We have defined the scope of the problem, ADS-B reports have been decoded, the output of the decoder has been dumped into a temporary file that has been imported by a python script and all the direct and indirect features have been obtained and assigned as attributes of a class called instance.

At the end of the program we added a little part to create the matrix and once it was fully operative, we had to choose data to run the code. The initial idea for this project was to use real data from 2020 but due to the COVID-19 outbreak and the dramatic reduction in air travel it was impossible to keep up with it. Instead, we used ADS-B reports obtained from the same antenna during 2017 which were stored in a server called Bleriot.

Four different months were chosen for the project and all the flights during that period were transformed into instances (rows) with their corresponding attributes or features (columns). The selected months were: January, February, July and August of 2017. These months were selected to introduce diversity in the data as two of them correspond to the winter season and the other two to the summer season. These months have very different meteorological conditions, different traffic densities and even different aircraft as some airlines change them depending on the time of the year. The matrix that we created had a size of 25,294 rows and 21 columns (19 numerical plus 2 categorical). Fig. 2.18 shows the result with some attributes.

	A	B	C	D	E	F	G	H	I	J	K	L
1	Time_to_land	Initial_time	ARP_distance	ARP_azimuth	Heading	Altitude	IAS	GS	Barometric_pressure	Wind_direction	Wind_variability	Wind_speed
2	2773	72011	67945.11919	79.60269227	242.0281858	3055.62	270	430	1018	220	0	5
3	1465	63166	67847.84333	10.90380796	23.56925828	4343.4	271	396	1018	210	35	10
4	2133	33490	67838.85632	79.76764726	221.3315809	4655.82	289	462	1020	200	45	8
5	2112	56126	67915.54812	77.1080515	214.2027516	3055.62	265	442	1018	210	0	11
6	1735	33488	67868.22098	96.0751405	277.4480894	3512.82	298	346	1020	200	45	8
7	1824	51949	67945.49319	45.43119121	179.2399269	3619.5	272	396	1018	210	0	12
8	2247	71076	67832.6822	79.77110112	210.646372	2887.98	263	440	1017	230	0	6
9	2425	33490	67791.68245	79.7918701	213.9627396	4960.62	261	424	1020	200	45	8
10	1465	54847	67962.86801	28.80005234	162.6966622	3528.06	247	460	1018	210	35	12
11	2294	71085	67905.0893	76.25520182	198.2382951	3139.44	258	428	1017	230	0	6
12	1042	57613	67946.11634	28.84796023	147.0560283	3489.96	281	396	1018	210	30	11
13	2079	71607	67954.87968	58.31519631	224.1070713	3177.54	272	446	1017	230	0	6
14	1225	44916	64062.64921	93.15421057	86.05232321	5250.18	250	362	1019	210	0	13
15	1625	53551	67910.26388	48.23409213	182.1093808	2567.94	253	460	1018	200	0	12
16	1638	33489	67951.46841	79.7650587	231.0706911	3360.42	289	366	1020	200	45	8
17	1236	33490	67946.40768	79.5718213	226.4417188	3169.92	273	346	1020	200	45	8
18	2007	79914	67653.56877	178.9517892	172.4846228	5692.14	242	456	1018	220	0	3
19	2558	60612	67719.30228	33.63166599	52.0823277	5090.16	266	478	1018	210	30	11
20	2691	39959	67950.63966	79.6524221	279.5607829	3070.86	293	424	1019	200	30	11
21	2000	56422	67978.14159	59.19138959	211.7108189	3017.52	256	424	1018	210	0	11
22	2090	70454	67959.70628	72.721818	197.9592801	2461.26	259	408	1017	230	0	6
23	3021	52441	67968.18791	96.07571847	272.0565188	2735.58	253	424	1018	200	0	12
24	2378	62778	67447.12812	80.21370305	213.6626367	4495.8	274	456	1018	210	0	10
25	1344	33489	67944.84263	45.40034504	189.0444701	3246.12	247	302	1020	200	45	8
26	1230	48116	67890.31484	29.0245899	171.5364258	3459.48	275	458	1019	210	0	13
27	1779	59346	67670.68277	113.8449673	303.7864326	3200.4	279	334	1018	210	30	11

Fig. 2.18 Output matrix from the python code.

2.4.1 Matrix data tuning

The created matrix was correct but it had to be adapted to the format used by Scikit-Learn which is the python ML library that we will use. To adapt it, two different techniques known as label binarization and numerical pipelines can be used.

2.4.1.1 Label binarizer

The two categorical attributes in the project are the runaways and the WTC. These features are not “understood” by python ML models as they can only handle numerical inputs, thus the features have to be converted.

Label binarization is a technique which consists on going from a categorical attribute to a numerical one by creating as many numerical columns as labels were in the first place [12]. This can be easily visualized in Fig. 2.19 which shows the result of applying a label binarization to the runaways.

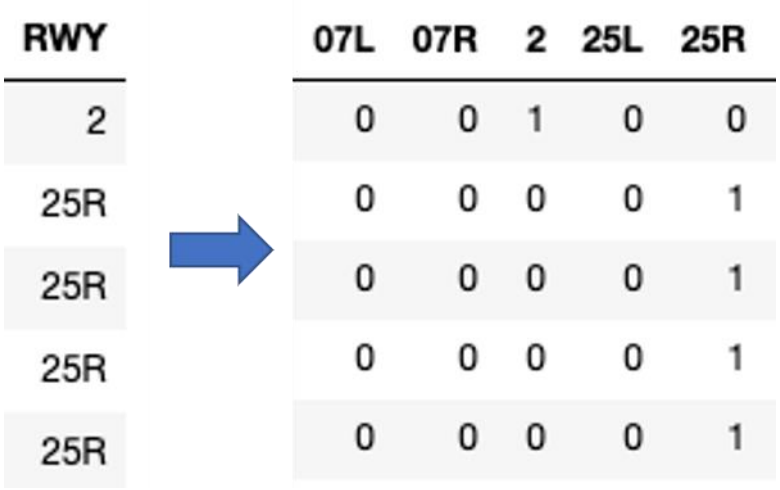


Fig. 2.19 Label binarization example for the runaways.

To do this conversion, we used a python class called LabelBinarizer [41]. Using it is particularly useful because not only it does the conversion but it also stores the information in a very efficient manner. Instead of storing a matrix with all the zeros and ones, it only stores the locations of the non-zero elements.

Label binarization could be used in this case because it only creates five columns for the runaways and four columns for the WTC. If the same technique had been used with the airlines (instead of dividing them in European, American, etc.) more than one hundred columns would appear flooding the dataset with useless information.

Other techniques such as classical encoders do not create extra columns but they present other problems. These encoders associate a number to each category, for example runaway 07L could be one, runaway 25L could be two, etc. The problem with this procedure is that ML algorithms may assume that two nearby values are more similar than two distant ones or that the highest value is better than the lowest one although this is completely false.

2.4.1.2 Pipelines

Pipelines are a sequence of data processing components. They are very common in ML since usually there is a lot of data to manipulate and a lot of transformations to apply. They allow a very robust architecture and the different elements in a pipeline do not interact between them. Each element pulls some data, processes it, and spits out the result in another data store [12].

In this case, as the dataset is not huge, there are not many transformations to do but it is always a good practice to use pipelines in case more data or extra features are to be added in the future. Pipelines were used for all the numerical features although the only function they contain is an imputer. Imputers are functions that replace any missing values of each attribute with “strategies”. The different strategies are: mean, median, most frequent and constant [42]. We decided to use the median strategy for the project.

Fig. 2.20 shows that before using the imputer, some features had missing items but after the pipeline they were filled and now all the features have the same number of non-null elements.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25294 entries, 0 to 25293
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Time_to_land         25294 non-null  int64
 1   Initial_time         25294 non-null  int64
 2   ARP_distance         25294 non-null  float64
 3   ARP_azimuth          25294 non-null  float64
 4   Heading              25294 non-null  float64
 5   Altitude             25294 non-null  float64
 6   IAS                  25291 non-null  float64
 7   GS                   25267 non-null  float64
 8   Barometric_pressure  25276 non-null  float64
 9   Wind_direction       25276 non-null  float64
 10  Wind_variability     25276 non-null  float64
 11  Wind_speed           25276 non-null  float64
 12  Visibility           25276 non-null  float64
 13  European_airlines    25294 non-null  int64
 14  American_airlines    25294 non-null  int64
 15  Latam_airlines       25294 non-null  int64
 16  Other_airlines       25294 non-null  int64
 17  Day_of_the_week      25294 non-null  int64
 18  Mix_index            25294 non-null  float64
 19  WTC                  25294 non-null  object
 20  RWY                  25294 non-null  object
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25294 entries, 0 to 25293
Data columns (total 28 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Initial_time         25294 non-null  float64
 1   ARP_distance         25294 non-null  float64
 2   ARP_azimuth          25294 non-null  float64
 3   Heading              25294 non-null  float64
 4   Altitude             25294 non-null  float64
 5   IAS                  25294 non-null  float64
 6   GS                   25294 non-null  float64
 7   Barometric_pressure  25294 non-null  float64
 8   Wind_direction       25294 non-null  float64
 9   Wind_variability     25294 non-null  float64
 10  Wind_speed           25294 non-null  float64
 11  Visibility           25294 non-null  float64
 12  European_airlines    25294 non-null  float64
 13  American_airlines    25294 non-null  float64
 14  Latam_airlines       25294 non-null  float64
 15  Other_airlines       25294 non-null  float64
 16  Day_of_the_week      25294 non-null  float64
 17  Mix_index            25294 non-null  float64
 18  H                    25294 non-null  int64
 19  J                    25294 non-null  int64
 20  L                    25294 non-null  int64
 21  M                    25294 non-null  int64
 22  07L                  25294 non-null  int64
 23  07R                  25294 non-null  int64
 24  2                    25294 non-null  int64
 25  25L                  25294 non-null  int64
 26  25R                  25294 non-null  int64
 27  landing_cat          25294 non-null  object
```

Fig. 2.20 Example of a numerical pipeline.

2.4.1.3 From time to land to categories

The last step in this chapter consists on transforming the time to land column into categories. To do so, we should first gain some insight with the data. The average time to land is 890 seconds, the highest value is 3,446 seconds and the lowest one is 254 seconds. Also, the majority of arrivals are distributed in a gap between 600 seconds and 1,200 seconds. This can be appreciated in Fig. 2.21 which shows the number of flights versus the time to land for the whole dataset.

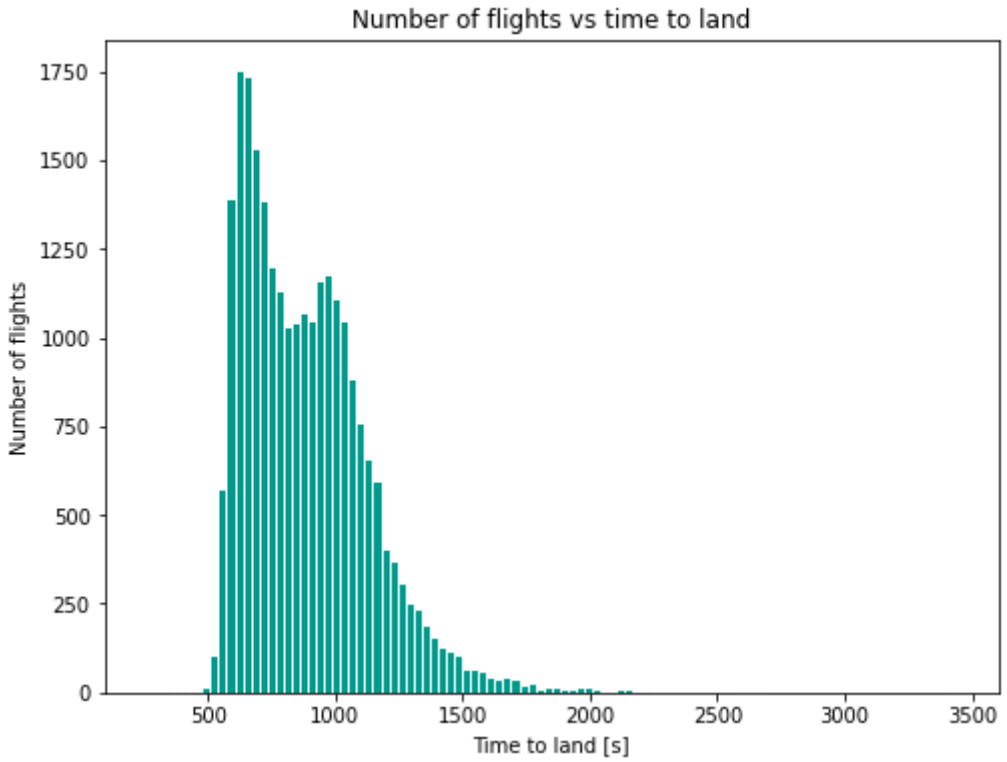


Fig. 2.21 Histogram showing the number of flights versus the time to land.

Regarding the categories, we can see that advanced flights are concentrated in a small bracket whereas the delayed ones are distributed in a period of more than 2,000 seconds. This makes sense since it is very difficult to save time during the approach as there are a lot of speed limitations but different factors can lead to severe delays and slow approaches.

To create the 3 different categories the criterion shown in Table 2.2 was applied, where the planned cases represent the 60% of all flights and the advanced and delayed ones represent a 20% each.

Table 2.2 Criterion to create three categories.

Category	Percentage (%)	Fastest flight [s]	Slowest flight [s]	Number of flights
Advanced	20	254	658	5058
Planned	60	658	1076	15176
Delayed	20	1076	3446	5058

Fig. 2.22 shows the histogram representing the different categories where the green bins are for the planned cases and the orange ones are delayed or advanced arrivals.

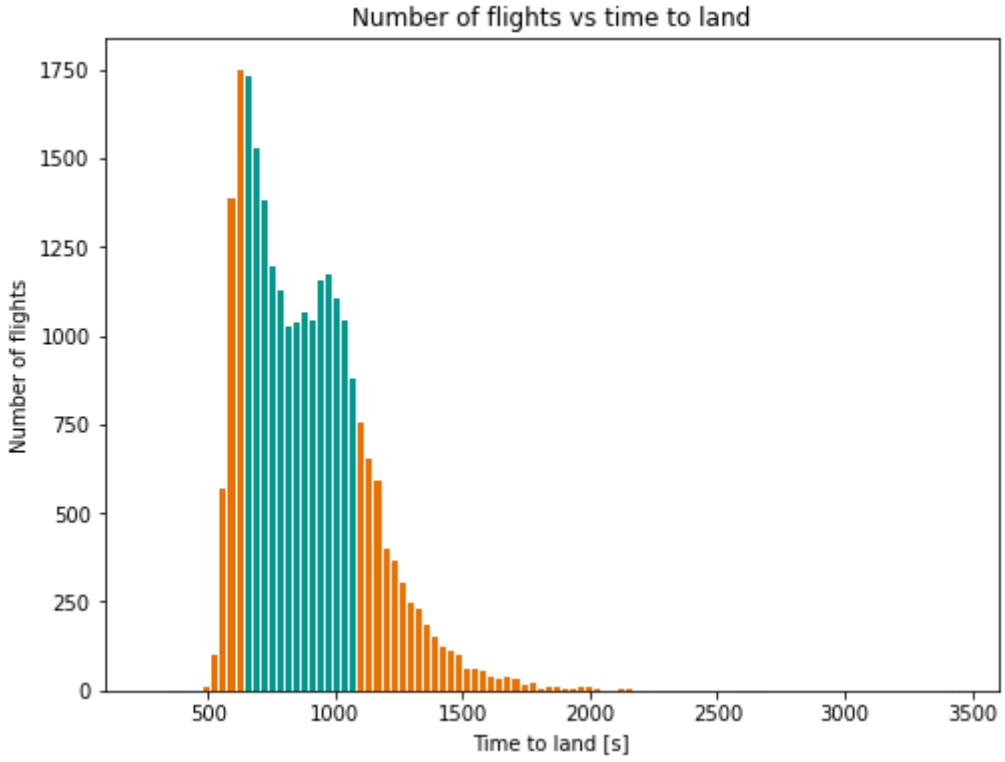


Fig. 2.22 Histogram for the three categories classification.

A very similar criterion was followed for the five categories problem, the only difference in this scenario is that the fastest and the slowest 2% where separated from their previous category to create new ones. The result can be seen in Table 2.3.

Table 2.3 Criterion to create five categories.

Category	Percentage (%)	Fastest flight [s]	Slowest flight [s]	Number of flights
Very advanced	2	254	565	506
Advanced	18	565	658	4553
Planned	60	658	1076	15176
Delayed	18	1076	1538	4553
Very delayed	2	1538	3446	506

Again, in an identical way, the histogram in Fig. 2.23 shows the number of flights versus the time to land but this time red bins are added to point out the extreme cases.

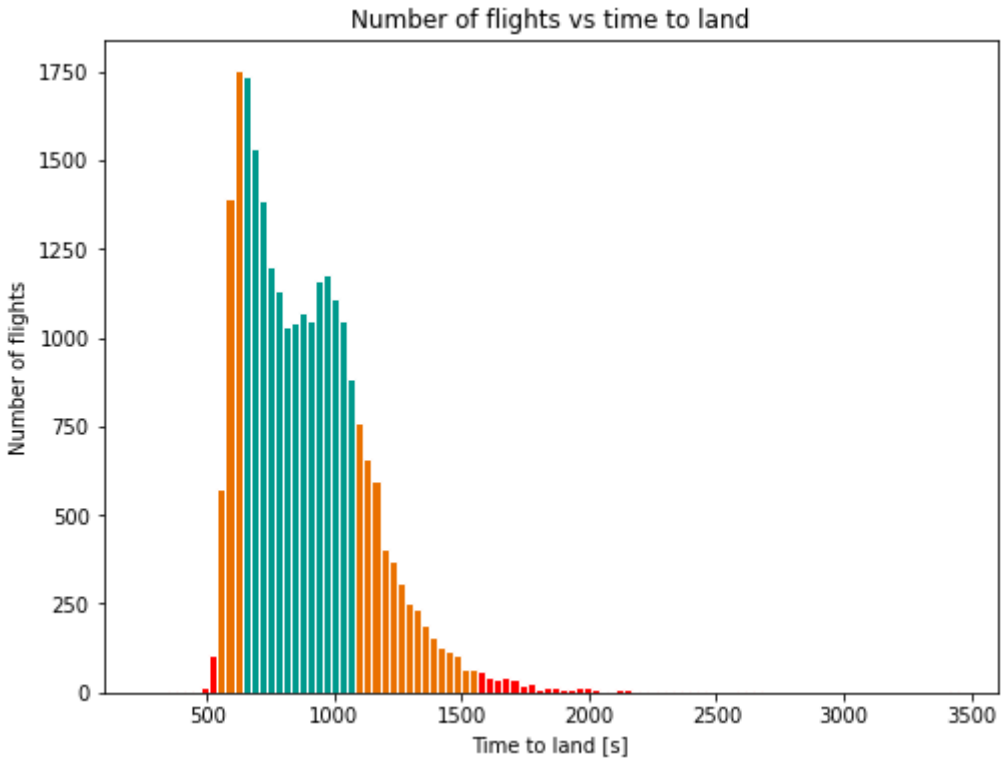


Fig. 2.23 Histogram for the five categories classification.

After defining the categories, a new column called landing categories was created and the time to land column was dropped from the input matrix. Now, we are ready to start training ML models.

CHAPTER 3. RESULTS

This chapter, contains the results of the project after training and testing the different ML models. The process that we followed to obtain the results is summarized in Fig. 3.1 and it is composed of the following steps:

1. The dataset is split into a training and a test set using either a stratified sampling or a train test split.

2. Six classifiers are evaluated using a CV of ten folds without any tuning and the best splitting strategy is chosen.

3. The hyperparameters of the three most promising models are tuned using a grid search and a random search.

4. The tuned estimators are boosted and combined to create ensemble methods.

5. The best model is tested for the five categories problem and oversampling techniques are used.

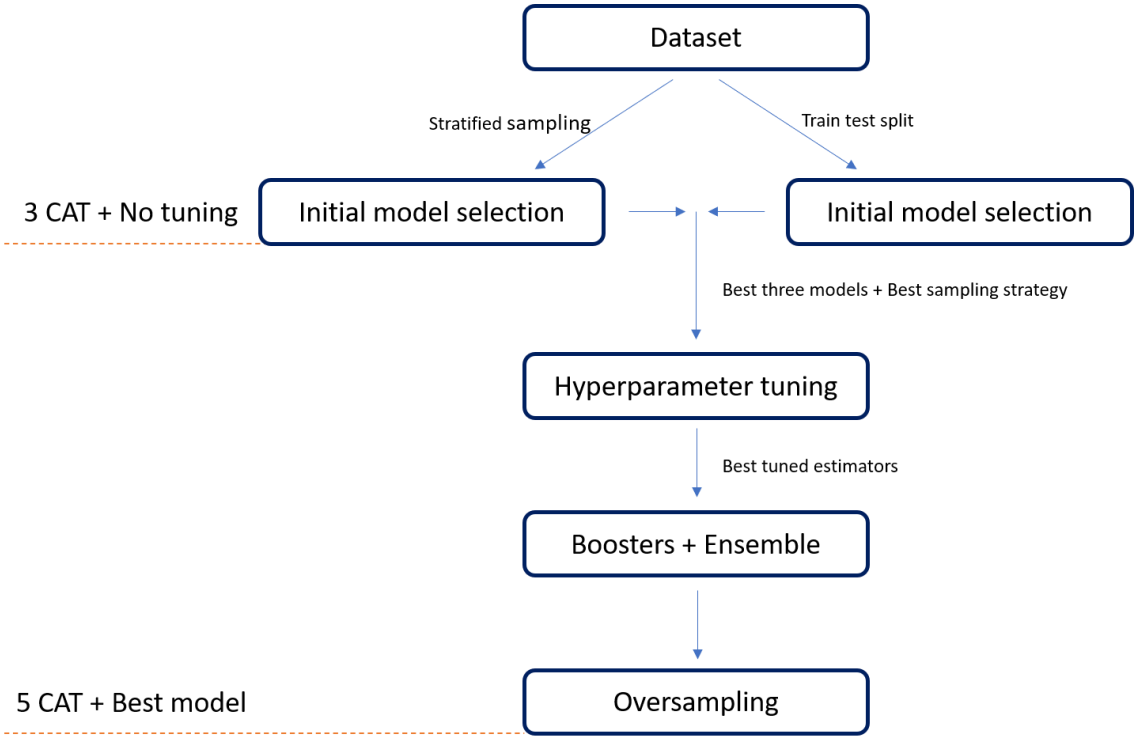


Fig. 3.1 Results obtention scheme.

It is worth mentioning that the different tables containing the results have a common column both for accuracy and recall although we said that they are different metrics. This happens because accuracy is in fact a particular case of

recall when it is calculated with a weighted strategy, which means that the recall on each category is averaged according to the number of instances in the category versus the total number of instances. If a different strategy had been chosen, it would result in different values of accuracy and recall [43].

3.1 Initial model selection

As it was explained in 1.5, tuning the hyperparameters is a very resource intensive process. Thus, we must obtain three models that are already promising before any kind of tweaking.

To assess the different options, we used a CV of ten folds both for the stratified sampling and the train test split. The scores shown in the different tables are obtained from the test set used on each fold. Additionally, the best model is the one with the highest f1 score.

Table 3.1 shows the performance measures for a stratified sampling strategy whereas Table 3.2 shows the results for a train test split.

Table 3.1 Stratified sampling initial results with no tuning.

Model	f1	Precision	Accuracy/ Recall	Best category
Random forest	0.7313	0.7430	0.7447	Planned
MLP	0.6746	0.6753	0.6820	Planned
Logistic regressor	0.6003	0.6180	0.6344	Planned
K-Nearest neighbors	0.6246	0.6239	0.6323	Planned
SVC	0.6197	0.6599	0.6604	Planned
Naive-Bayes	0.2391	0.5250	0.3204	Advanced

Table 3.2 Train test split initial results with no tuning.

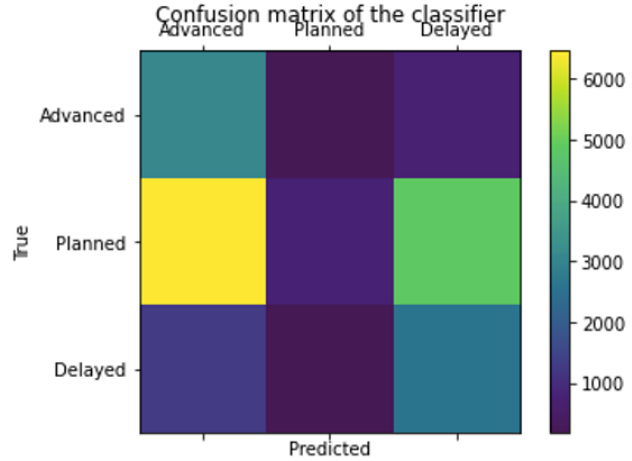
Model	f1	Precision	Accuracy/ Recall	Best category
Random forest	0.7249	0.7363	0.7398	Planned
MLP	0.6669	0.6704	0.6796	Planned

Logistic regressor	0.5970	0.6142	0.6327	Planned
K-Nearest neighbors	0.6207	0.6194	0.6283	Planned
SVC	0.6177	0.6640	0.6623	Planned
Naive-Bayes	0.2489	0.5190	0.3225	Advanced

From the above tables, the first conclusion that we can obtain is that generally a stratified sampling works better than a train test split. Hence, the grid search and the random search were performed on a dataset split with a stratified sampling strategy.

Regarding the ML models, it can be seen that the random forest classifier and the MLP are clearly the best ones. On the other hand, to select a third model, there is a “tie” between the SVC and the K-nearest neighbors as both of them present very similar figures. We decided to run an extensive hyperparameter search on the SVC as it has a higher precision and accuracy but also a fast one on the K-nearest neighbors to see how it responded without sacrificing much time.

Finally, another fact that draws attention is the behaviour of the Naive Bayes classifier. While all models seem to perform best on the planned category, this classifier performs best on the advanced category. Also, all of them are achieving f1 and accuracy scores of at least 60% and this one is struggling to get to 30%. To understand better what is happening, we can look at Fig. 3.2 which shows the confusion matrix for this case.




```
Confusion matrix: [[3104  207  750]
                   [6466  790 4873]
                   [1269  185 2591]]
```

Fig. 3.2 Confusion matrix of a Naive Bayes classifier.

We can see that the model is actually really good at predicting advanced instances as it reaches an accuracy of 76% and it performs reasonably well with the delayed category reaching an accuracy of 64%. The problem comes with the planned category as only a 6.5% of the instances in that category are correctly predicted although they represent 60% of the flights. Generally, we could say that the model has a strong tendency to favour the less represented cases.

The reasons for this behaviour are probably three:

1. The different instances are not really independent although the model is making that assumption.

2. The version that we used is called ComplementNB and it is particularly suited for unbalanced datasets. This favours the advanced and delayed categories as there are less samples.

3. Naive Bayes models work better when the variables of the problem are discrete but in this case there a lot of continuous variables.

3.2 Fine tuning the models

3.2.1 Best three models

The tables on this section show the different hyperparameter combinations that we tried and their optimal values for a grid search and a random search. The performance measures were obtained by fitting the best estimator with the training data and predicting on the test data.

Table 3.3 shows the optimal hyperparameters for a random forest classifier and Table 3.4 shows the performance measures obtained from the best estimator.

Table 3.3 Hyperparameter tuning results for a random forest classifier.

Hyperparameter	Possible values	Grid search optimal value	Random search optimal value
Number of estimators	100/200/500/1000/200	500	500
Bootstrap	True, False	False	False
Class weight	Balanced, Balanced subsample, None	Balanced	Balanced subsample
CCP alpha	0/0.5/1.5	0	0

Minimum samples per leaf	0.5/1/1.5	2	2
Max features	Auto, Sqrt, Log2	Log2	Log2
Criterion	Gini, Entropy	Gini	Gini

Table 3.4 Performance measures for the best random forest estimator.

Performance measure	Grid search best result	Random search best result
f1	0.7493	0.7506
Precision	0.7487	0.7498
Accuracy/Recall	0.7527	0.7535

Table 3.5 shows the optimal hyperparameters for an MLP classifier and Table 3.6 shows the performance measures obtained from the best estimator.

Table 3.5 Hyperparameter tuning results for an MLP classifier.

Hyperparameter	Possible values	Grid search optimal value	Random search optimal value
Hidden layer sizes	50/75/100/200	75	100
Activation	Identity, Logistic, Tanh, Relu	Logistic	Tanh
Solver	Lbfgs, Sgd, Adam	Lbfgs	Adam
Learning rate	Constant, Invscaling, Adaptive	Constant	Constant
Power t	0.3/0.5	0.5	0.5
Shuffle	True, False	False	False
Initial learning rate	1e-3/5e-4	1e-3	1e-3

Table 3.6 Performance measures for the best MLP estimator.

Performance measure	Grid search best result	Random search best result

f1	0.6857	0.6882
Precision	0.6868	0.6884
Accuracy/Recall	0.6948	0.6946

Table 3.7 shows the optimal hyperparameters for an SVC classifier and Table 3.8 shows the performance measures obtained from the best estimator.

Table 3.7 Hyperparameter tuning results for an SVC classifier.

Hyperparameter	Possible values	Grid search optimal value	Random search optimal value
Class weight	Balanced, None	None	None
Gamma	Scale, Auto	Auto	Auto
Kernel	Linear, Poly, Rbf, Sigmoid	Poly	Poly
C	0.5/1/1.5/2/3	3	2
Shrinking	False, True	True	True
Coef0	-1/0/1	1	1
Degree	2/3/5	5	5
Tol	1e-3/1e-4	1e-4	1-3
Break ties	False, True	True	True

Table 3.8 Performance measures for the best SVC estimator.

Performance measure	Grid search best result	Random search best result
f1	0.6594	0.6565
Precision	0.6596	0.6569
Accuracy/Recall	0.6687	0.6667

In the light of the above, the random search and the grid search present very similar performances but their execution times are radically different. A good random search lasts more or less a couple of hours depending on the number of iterations that we want to try but a normal grid search will last at the very least one day. In fact, some algorithms like the MLP can last more than a week as they do not admit parallelization. Given that, we can affirm without any doubt that the random search is the best option as we will obtain the same results investing less

time and computing capacity. The only situations where it would be reasonable to use a grid search is if we had either very powerful machines or a lot of time.

Fig. 3.3 shows the improvement in the different performance measures that we accomplished. For instance, the lighter colours indicate the value before tuning and the darker ones the values after the search.

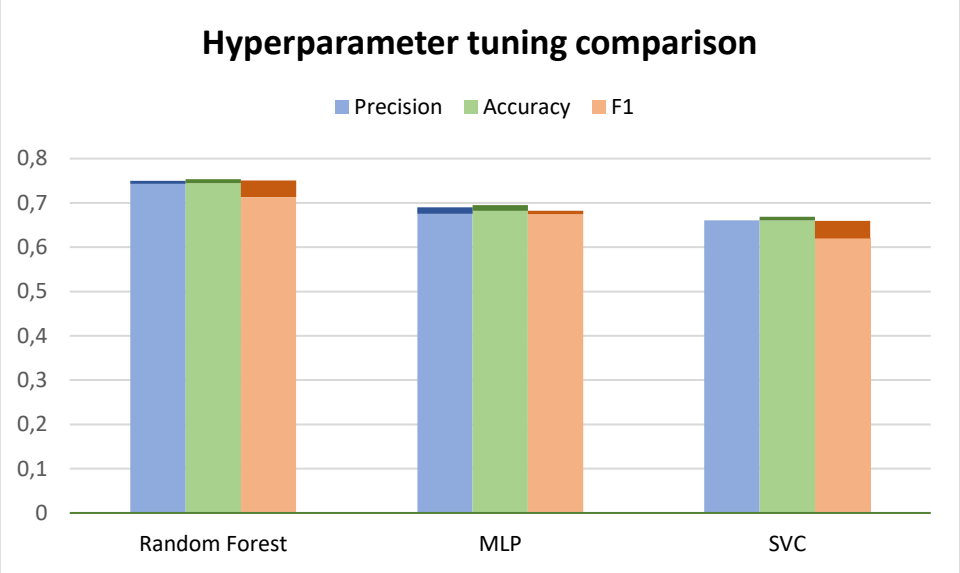


Fig. 3.3 Performance measures of the different estimators.

Finally, it is clear that the random forest classifier is the best model so far. Hence, we decided to try the boosters and the five categories problem with the best random forest estimator.

3.2.2 K-Nearest neighbors

As we also wanted to try how the K-Nearest neighbors improved with a fast hyperparameter tuning, we ran a short random search on the algorithm and the best values were found to be the ones in Table 3.9 improving its f1 score by a 4.4%, its accuracy by a 4.69% and its precision by a 4.77%.

Table 3.9 Hyperparameter tuning results for a K-Nearest neighbors classifier.

Hyperparameter	Possible values	Random search optimal value
Leaf size	10/20/30	10
Number of neighbors	3/5/10	10

CHAPTER 2.Weights	Uniform, Distance	Distance
P	1/2/3	1

3.3 Boosting and ensemble methods

Regarding the boosting techniques, the performance of the AdaBoostClassifier was quite disappointing as it did not improve the results of the random forest. In fact, all the performance measures went down by approximately a 1%. If boosters were used in binary classification problems such as predicting if a time to land is going to be as planned or not, the results would probably get better.

On the other hand, the ensemble methods were able to improve the performance of the random forest but the results were far from excellent. This method increases the maximum accuracy by a 0.5% and the f1 score and the precision by around a 0.25% each. Although these figures represent an increase, it is not a substantial one.

3.4 Five categories and oversampling

We also tested how the random forest performed with the five categories problem and if by means of oversampling techniques we were able of improving the results of the extreme cases. Table 3.10 shows the performance measures of the training set with a ten folds CV before and after implementing oversampling techniques. The very advanced and very delayed categories which originally contained 506 instances each, were oversampled so that 5,000 flights appeared on each one.

Table 3.10 Performance measures for the five categories problem.

Performance measure	No oversampling	Oversampling
f1 score	0.7071	0.6866
Precision	0.7092	0.7005
Accuracy	0.7180	0.7095

We can easily notice that the overall results are worse after the oversampling and even though it may seem counterintuitive this is something good. When we increased the number of samples on the extreme cases, we were making it more difficult for the classifier to correctly make predictions as some bias and noise was being introduced. In turn, the recall of the very advanced and very delayed categories could improve and this is what actually happened. Fig. 3.4 shows that the recall of the very delayed category was more than doubled going from a 16% to a 34%, on the other hand, the recall of the very advanced category went from 26% to 35%. Thus, we can claim that oversampling techniques worked perfectly ad they allow us to vastly improve the results of the underrepresented classes.

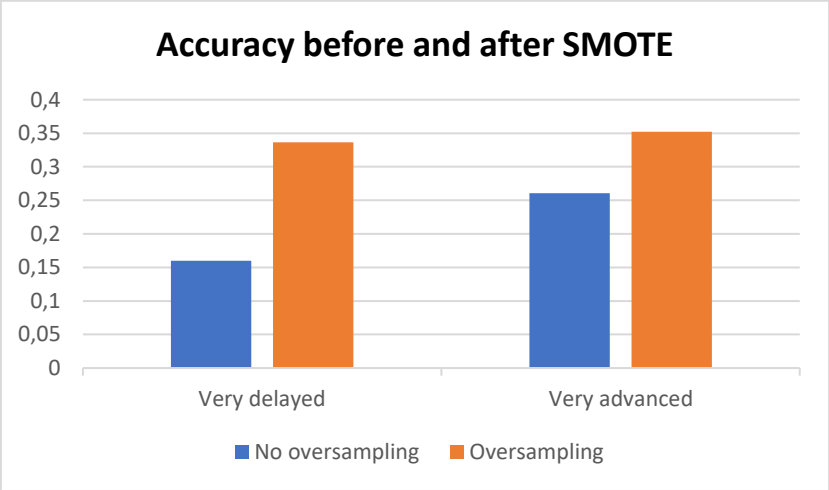


Fig. 3.4 Accuracy change in the extreme cases before and after SMOTE.

3.5 Final assessment

During the different tests, we accidentally came across a very interesting result. The confusion matrix on the right-hand side of Fig. 3.5 can be obtained with an SVC classifier tuned with the following hyperparameters: gamma set to scale, class weight set to balanced and kernel set to rbf.

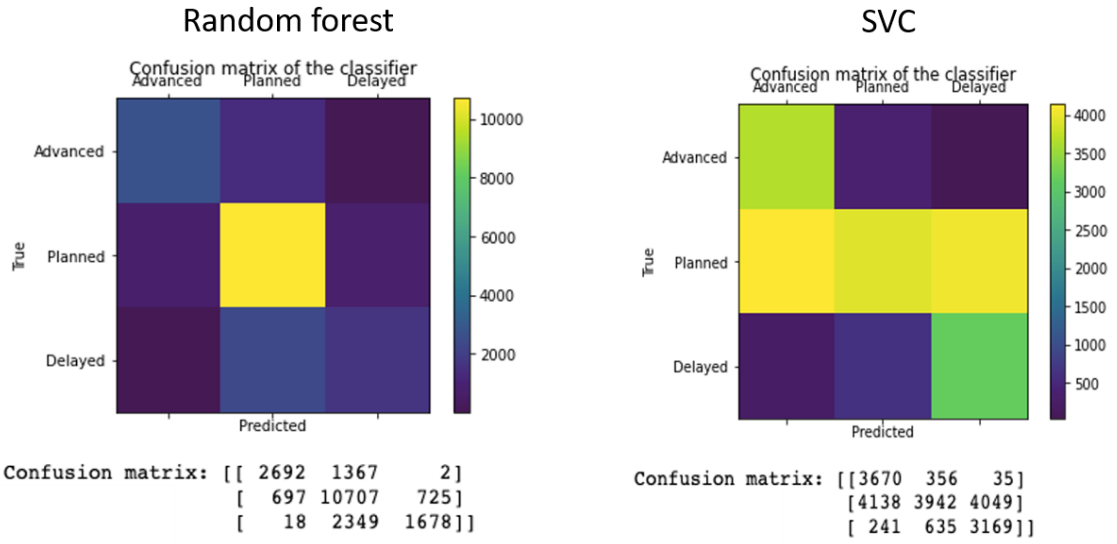


Fig. 3.5 Comparison of the confusion matrix of a random forest and an SVC classifier.

Although this SVC solution is not optimal, in fact its f1 score is only a 50% (16% less than the best one), the results for the advanced and delayed categories are really good. The accuracy of the advanced flights is a 90% and for the delayed ones, we reach a 78%. On the contrary, the accuracy of the planned flights is as low as a 32.5%. Anyway, when we look at the left-hand side of Fig. 3.5 we can see that the accuracy of the planned flights with a random forest is around 88%.

The conclusion is: the best way to evaluate if a landing time in Barcelona-El Prat airport is going to be as planned, advanced or delayed is to use an SVC classifier and a random forest. We feed an instance to both models and we ask them to make a prediction but also to calculate the probability of that prediction being correct with the sklearn function predict_proba. If they agree, we directly classify the instance and if there is a disagreement between them, we use the probability function to break the tie.

CONCLUSIONS

This project presents a method to calculate the time to land in Barcelona-El Prat airport. To do that, ADS-B data was decoded, processed with a python script and arranged in a matrix format that could be used as an input for ML classification models.

Thanks to a function called features importance, we have been able to assess how the different variables contribute to the final decision of the classifiers. The most important ones are the ARP azimuth and the altitude followed by the initial time, the heading and distance to the ARP. Two hypotheses were also made claiming that neither the airline type nor the day of the week would have a strong impact on the decision process. Whereas the first one is fulfilled, indicating that ATC do not present any kind of preference for certain airlines, the second one is not. It was found that the day of the week has indeed an intermediate influence on the classifiers above other features such as the runaways or the visibility.

Two different strategies to divide the test and the training set were tried and it was found that a stratified sampling strategy delivers better results. This is consistent with the idea that the data from which models learn, must be unbiased and must be as representative of the test data as possible.

Regarding the different ML models, the Naive Bayes classifier was the one with the poorest results. On the other end, we find the random forest classifier, despite its simplicity, the results are outstanding. If we compare it with the second-best classifier (the MLP), we can see that an untuned random forest already offers better results. Looking at classifiers with an intermediate performance it is interesting to see the difference between the SVC and the logistic regression classifier. The SVC brings slightly better results and this is probably because although they use similar algorithms, the first one looks for the hyperplane which divides the variables and the second looks one only for a line. Hence, in highly nonlinear problems like this one, containing 27 features it is more reasonable to go for the hyperplane.

The hyperparameter tuning has proven to be very effective, specially the random search, since it allowed us to improve the results in a reasonable time and without investing an excessive amount of resources. Also, the oversampling techniques allowed us to enhance the less represented categories by creating new instances from the existing ones. Perhaps combining this with an under sampling of the most represented classes, would bring even better results.

Finally, we concluded that to correctly predict the time to land in Barcelona, the most reasonable strategy is to use a combination of an SVC and a random forest classifier along with a confidence score of their predictions.

The future work of this project should be mainly focused on further improving the reliability of the ML models. To do so, the most logical idea would be to increase the data volume by increasing the number of instances and features of the dataset or trying new models that offer better results.

Adding more instances is relatively easy as the server already contains all the flights that landed in Barcelona during 2017 and the two final months of 2016. As the code to convert ADS-B reports into input matrices is already created, we should only process the data and reproduce again the results to see if something changes.

Adding features is more complicated because new code has to be developed but in turn it could be much more effective than just increasing the instances. Some unknown variables could perhaps be the key to better estimators. Three interesting features that could be added are: the number of airplanes in the airspace at the same time (traffic density), the cumulative delay of the airport when the airplane crosses the circle and a binary category indicating if the plane has been doing any kind of holding before entering the circle.

Another idea for future work would be to completely redirect the scope of the problem. Instead of predicting the time to land, we could predict what are the waypoints that the different planes will fly over in their approaches towards Barcelona. This is also a classification problem that could be tackled by creating one classifier per waypoint able to recognize if a plane is going to fly over it or not.

BIBLIOGRAPHY

[1] Microsoft, “Artificial Inteligence,” 2019. [Online]. Available: https://docs.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model#:~:text=A%20machine%20learning%20model%20is,and%20learn%20from%20those%20data.&text=See%20Get%20ONNX%20models%20for%20Windows%20ML%20for%20more%20information..

[2] Scikit learn, “Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.,” 2020.

[3] Wikipedia, “Pandas (software),” 2020. [Online]. Available: https://en.wikipedia.org/wiki/Pandas_(software).

[4] Wikipedia, “NumPy,” 2020. [Online]. Available: https://es.wikipedia.org/wiki/NumPy.

[5] Wikipedia, “Random Forest,” 2020. [Online]. Available: https://en.wikipedia.org/wiki/Random_forest.

[6] Scikiit learn, "Random Forest", 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[7] A. Chakure, “towardsdatascience.com,” 2019. [Online]. Available: https://towardsdatascience.com/random-forest-classification-and-its-implementation-d5d840dbead0.

[8] Wikipedia, “Logistic regression,” 2020. [Online]. Available: https://en.wikipedia.org/wiki/Logistic_regression.

[9] Scikit learn, “Logistic Regression,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[10] Primoai, “Logistic Regression,” 2020. [Online]. Available: http://primo.ai/index.php?title=Logistic_Regression_(LR).

[11] Deeplearning.net, “Tutorial MLP,” 2020. [Online]. Available: http://deeplearning.net/tutorial/mlp.html.

[12] A. Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, O'REILLY, 2017.

[13] Scikit learn, “MLP,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html?highlight=mlp%20classifier#sklearn.neural_network.MLPClassifier.

[14] Wikipedia, “K-nearest neighbors algorithm,” 2020. [Online]. Available: https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm#/media/File:KnnClassification.svg.

[15] aprendemachinelearning.com, “Clasificar con K-Nearest-Neighbor en Python,” 2018. [Online]. Available: https://www.aprendemachinelearning.com/clasificar-con-k-nearest-neighbor-ejemplo-en-python/.

[16] Scikit learn, “KNeighbors Classifier,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html.

[17] J. Amat, “cienciadedatos.net,” 2017. [Online]. Available: https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines.

[18] R. Gandhi, «towardsdatascience.com,» 2018. [En línea]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[19] Scikit learn, “SVC,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html.

[20] Scikit learn, “Naive Bayes classifier,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.ComplementNB.html#sklearn.naive_bayes.ComplementNB.

[21] faculty.elgin.edu, “Other effective sampling methods,” 2020. [Online]. Available: https://faculty.elgin.edu/dkernler/statistics/ch01/1-4.html.

[22] A. G. a. L. Capelo, Applied deep learning with python., Birmingham - Mumbai: Packt, 2019.

[23] Scikit learn, “MinMaxScaler,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[24] Scikit learn, “Standard scaler,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

[25] S. Geller, “towardsdatascience.com,” 2019. [Online]. Available: https://towardsdatascience.com/normalization-vs-standardization-quantitative-analysis-a91e8a79cebf.

[26] Scikit learn, “Cross validation,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/cross_validation.html.

[27] Scikit learn, “GridSearchCV,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.

[28] Scikit learn, “Randomized Search,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html.

[29] S. Mangale, “www.medium.com,” 2019. [Online]. Available: https://medium.com/@sanchitamangale12/voting-classifier-1be10db6d7a5.

[30] J. Brownlee, “www.machinelearningmastery.com,” 2016. [Online]. Available: https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/.

[31] M. Sunasra, “Performance metrics,” 2017. [Online]. Available: https://medium.com/@MohammedS/performance-metrics-for-classification-problems-in-machine-learning-part-i-

b085d432082b#:~:text=We%20can%20use%20classification%20performance,primarily%20used%20by%20search%20engines..

[32] Skybrary, “ADS-B,” 2020. [Online]. Available: https://www.skybrary.aero/index.php/Automatic_Dependent_Surveillance_Broadcast_(ADS-B).

[33] M. Aicart, «Taxi time analysis and prediction with ADS-B data. A case study in Barcelona-El Prat Airport.,» Castelldefels, 2017.

[34] AENA, «Tráfico de pasajeros, operaciones y carga en los aeropuertos españoles.,» 2017.

[35] Pilotaware, “Hex-ID,” 2019. [Online]. Available: https://pilotaware.com/knowledge-base/hex-id/.

[36] python.org, “Pickle,” 2020. [Online]. Available: https://docs.python.org/3/library/pickle.html.

[37] Planepics, “Planepics,” 2014. [Online]. Available: http://www.planepics.org/cms/index.php/guides/13-guides/europe/154-barcelona#:~:text=Runways%2FWinds%3A%20Barcelona's%20airport%20has,one%20crossing%20runway%2002%2F20.&text=With%20both%20configurations%2C%20only%20the,25L)%20is%20used%20for%20departures..

[38] Wikipedia, “METAR,” 2020. [Online]. Available: https://en.wikipedia.org/wiki/METAR.

[39] Skybrary, “Wake Turbulence Category,” 2020. [Online]. Available: https://www.skybrary.aero/index.php/ICAO_Wake_Turbulence_Category.

[40] FAA, 1983. [Online]. Available: https://www.faa.gov/documentLibrary/media/Advisory_Circular/150_5060_5_part1.pdf.

[41] Scikit learn, “Label Binarizer,” 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html.

[42] Scikit Learn, “Simple Imputer,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html.

[43] stackoverflow.com, “Averaging scores,” 2019. [Online]. Available: https://stackoverflow.com/questions/55740220/macro-vs-micro-vs-weighted-vs-samples-f1-score.

[44] J. Sun, “The 1090MHz Riddle,” 2017. [Online]. Available: https://mode-s.org/decode/book-the_1090mhz_riddle-junzi_sun.pdf.

[45] Scikit learn, “Random Forest Classifier,” 2020. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[46] S. Saxena, “Precision vs Recall,” 2018. [Online]. Available: https://towardsdatascience.com/precision-vs-recall-386cf9f89488.

APPENDICES

APPENDIX A. Code structure

The data preparation program called csv2col, takes as an input the ADS-B data previously obtained from the ADS-B decoder (CSV temporary file) and it ends up generating a matrix that will be used as in input for ML models.

The program has a class called Instance and each attribute of this class is a feature. The code loops over all the aircraft in the ADS-B temporary file and it creates as many class objects as ICAO hexadecimal codes it can find. The attributes of every object are filled with different functions. Some attributes, like the runaways, have associated functions that only fill them. On the other hand, there are functions that can compute multiple attributes.

After filling all the attributes, an instance cleaner runs over all the class objects and it eliminates those with incorrect or missing data.

Finally, a matrix containing one flight per row and one feature per column is generated. This matrix is also a CSV file.

Fig. A.1 shows the scheme of the data processing program.

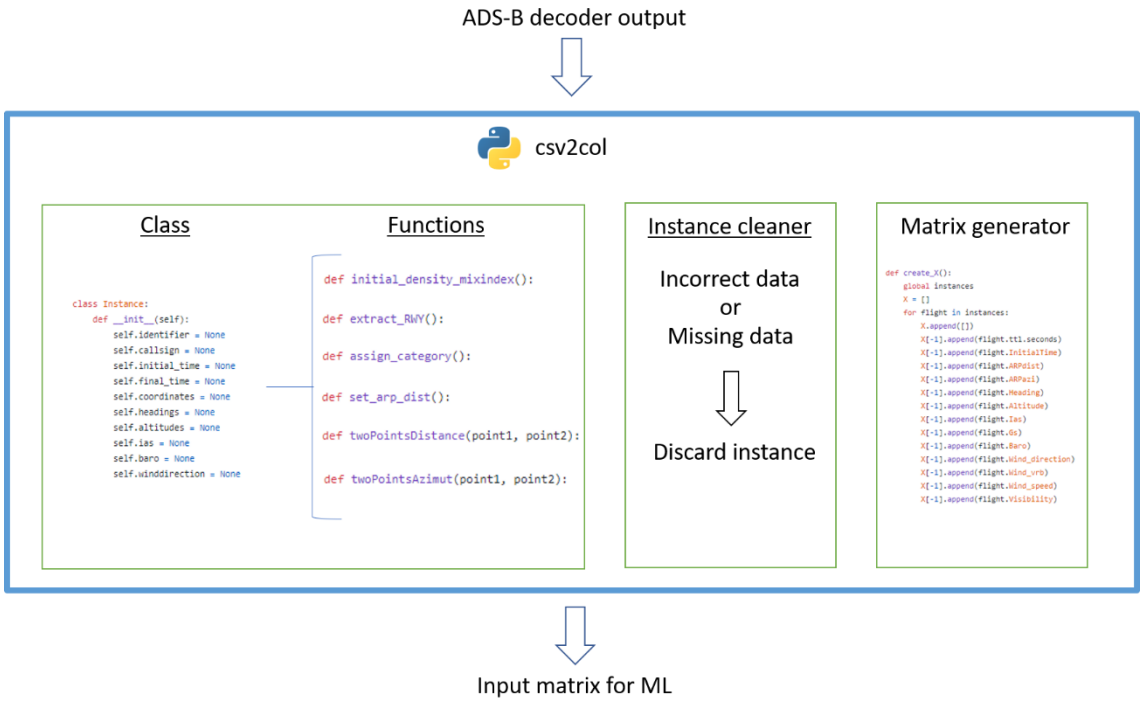


Fig. A.1 Scheme of the data processing program.